NPS52-87-020

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A SIMULATION STUDY OF A SPEED CONTROL SYSTEM FOR AUTONOMOUS ON-ROAD OPERATION OF AUTOMOTIVE VEHICLES

by

Michael J. Dolezal

June 1987

Thesis Advisor: Robert B. McGhee

Prepared for:

United States Army Combat Development Command
Fort Ord, CA 93941

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

D. A. Schrady
Provost

This thesis is prepared in conjunction with research sponsored in part by contract from the United States Army Combat Developments Experimentation Center (USACDEC) under MIPR ATEC 88-86.

The issuance of this thesis as a technical report is concurred by:

URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION unclassified	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
DECLASSIFICATION/DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-87-020	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION val Postgraduate School	6b OFFICE SYMBOL (If applicable) 52
	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
ADDRESS (City, State, and ZIP Code) nterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Com- t Developments Experimentation Center	8b OFFICE SYMBOL (If applicable) MIPR ATEC 88-86
ADDRESS (City, State, and ZIP Code) ort Ord, California 93941-5012	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

TITLE (Include Security Classification)
A SIMULATION STUDY OF A SPEED CONTROL SYSTEM FOR
UTONOMOUS ON-ROAD OPERATION OF AUTOMOTIVE VEHICLES

PERSONAL AUTHOR(S)
Dolezal, Michael J.

TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
ster's Thesis	FROM _____ TO _____	1987 June	270

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	simulation study; speed control; autonomous vehicles

ABSTRACT (Continue on reverse if necessary and identify by block number)

he study of human driving of automotive vehicles is an important aid to he development of viable autonomous vehicle navigation and control echniques. Observation of human behavior during driving suggests that his activity involves two distinct levels, the conscious and the un-onscious.

he behavior of a driver while stopping his vehicle at a stop sign can e conscious or unconscious, depending on the driver's skill level and he driving conditions. The driver's behavior involves a difficult rocess of estimating the distance to the stop sign and the velocity f the vehicle. Using these estimates, the driver then takes the ecessary control actions to stop the vehicle. This research attempts o mimic the driver's conscious and unconscious behavior through mathe-atical modeling and computer simulation.

DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION unclassified
NAME OF RESPONSIBLE INDIVIDUAL rof. Robert B. McGhee	22b TELEPHONE (Include Area Code) (408) 646-2095 / 22c OFFICE SYMBOL Code 52Mz

FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
unclassified

Approved for public release, distribution is unlimited

A Simulation Study of a Speed Control System for Autonomous On–Road Operation of Automotive Vehicles

by

Michael J. Dolezal
Major, United States Marine Corps
B.S., St. John's University, 1970

Submitted in partial fulfillment of the
requirements for the degree(s) of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1987

ABSTRACT

The study of human driving of automotive vehicles is an important aid to the development of viable autonomous vehicle navigation and control techniques. Observation of human behavior during driving suggests that this activity involves two distinct levels, the conscious and the unconscious.

The behavior of a driver while stopping his vehicle at a stop sign can be conscious or unconscious, depending on the driver's skill level and the driving conditions. The driver's behavior involves a difficult process of estimating the distance to the stop sign and the velocity of the vehicle. Using these estimates, the driver then takes the necessary control actions to stop the vehicle. This research attempts to mimic the driver's conscious and unconscious behavior through mathematical modeling and computer simulation.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would especially like to acknowledge the assistance and encouragement that my advisor, Dr. Robert B. McGhee, gave me throughout this research. He provided the opportunity to pursue a subject that is both interesting and challenging. His guidance, patience, and advice contributed substantially to the success of my research. I am extremely grateful to him for his time and counseling.

My appreciation also goes to Dr. Michael J. Zyda. His suggestions and comments, as second reader, have added significantly to the accuracy and readability of this thesis. Additionally, he has provided me with the necessary background and techniques to build the graphics simulation model used in this work.

I would also like to thank Chiam Huat Tan, who developed the vehicle simulation used as a basis for this work. He contributed a number of hours answering questions I had about graphics, the vehicle simulation, and the "C" programming language.

Lastly, I would like to thank my family, Joanne, Joey, and Rosemary. It is difficult to complete a Masters program under the best of circumstances, but it is impossible without a very understanding, helpful, and loving wife. Thank you for your support.

I. INTRODUCTION

A. GENERAL BACKGROUND

Man has been intrigued by the concept of robots for centuries. The earliest robots were mechanical toys moved by clockwork mechanisms. As time passed, the world became more complex and man began to develop concepts which would later provide a basis for research and experimentation in more versatile robotics. Jules Verne, an author of science fiction, was one of those pioneers. He somehow anticipated a most successful method in robotics, the use of pneumatic or hydraulic actuators for individual joints in a steam elephant. [Ref. 1]

Over the years, the concept of a robot has changed considerably. Science fiction writers and movie makers often project the image of a robot as some type of sublime creature or menacing evil. The popular American idea is that of an artificial man [Ref 2]. The concept of an artificial man is linked to the belief that mechanical slaves could free a substantial portion of the world's population from manual work [Ref. 3].

While not humanlike in appearance, in fact, mechanical slaves called industrial robots have been developed for production work. These robots normally operate from a fixed location and are programmed to do tedious and repetitive tasks. Typically, the programming is accomplished by either leading the robot through the desired movements and recording these movements, or by coding a

path consisting of sequences of linear and circular motions [Ref. 4:p. 4]. Recently however, special robot programming languages have been developed which, when combined with modern force and torque sensors, allow the robot to adjust to some variations in the environment [Ref. 4:pp. 395-410].

The next logical step in robot development is some type of mobile robot. The "automated guided vehicle" provides an intermediate level between fixed robots and unconstrained mobility. Here, navigation problems are avoided by using a control network which may be a wire, painted stripe, or track on a factory floor [Ref. 4:p. 8].

Research is continuing toward the idealized and unconstrained robot. This human-like or autonomous robot is capable of making decisions and adapting to environmental changes which may affect its purpose. In order to adapt to its environment, the autonomous robot requires numerous human-like sensors for input. Thus, studies are being conducted in the areas of vision, touch, and hearing. Additionally, research is ongoing in related areas of artificial intelligence. The objective is an autonomous robot capable of analyzing sensor input and making decisions to produce intelligent actions [Ref. 4]. Currently, studies of such machines are very diversified and must be integrated at some time in the future to produce an effective autonomous robot.

Perhaps then, the way to construct an autonomous robot capable of decision making in real-world problems is to integrate the features of a human; the human senses, the human brain, and human behavior. This would result in a very

complex hierarchical system with interactive and parallel processors. [Ref. 5] A system of this type is well beyond the scope of this work. Therefore, the objective of this work is limited to the study of a small portion of human behavior. Specifically, this research investigates and attempts to mimic the mental process by which a human driver controls the speed of a conventional automotive vehicle when coming to a stop at a stop sign or a traffic light. This is an area of autonomous vehicle research which has been largely ignored, but which may well be pertinent to the viability of future autonomous vehicles, especially on-road and wheel-based vehicles.

B. ORGANIZATION

Chapter II introduces vehicle dynamics and discusses human control of speed and direction in vehicles. Additionally, this chapter reviews a number of research projects relating to longitudinal speed control in autonomous vehicles. Finally, Chapter II introduces computer vision and discusses its limitations.

The objective of this research work in relation to autonomous vehicles and human drivers is discussed in greater depth in Chapter III. In that chapter, the assumptions concerning conventional automotive vehicle mechanics and characteristics of human driving are detailed. Those assumptions are described to show that the graphics simulation implemented for this study ignores many of the complex interactions that occur between a human driver, his vehicle, and the environment while traveling on the highway. These assumptions are made to

make the graphics simulation manageable and feasible within the time constraints of the study. The mathematical model of longitudinal speed control by a human driver used in this research is derived and detailed in Chapter III.

In Chapter IV, the design and implementation of the graphics simulation for the mathematical model are presented. The graphics workstations and the displays on each workstation are discussed in detail. Additionally, this chapter elaborates on the computer networking system and how it is used in this project. The overall design strategy and key issues of longitudinal speed control for the mathematical model are addressed in depth. The functions and implementation of the various modules developed for the simulation are described. Finally, a user's guide is included.

Numerous experiments are conducted with the graphics simulation to verify the mathematical model of Chapter III and validate this work. Chapter V records and explains the results of the experiments conducted using the simulation model developed in Chapter IV.

The last chapter, Chapter VI, summarizes the work and its potential benefit to autonomous vehicle research. Some suggestions concerning possible extensions to the research are also provided. Additional work in these areas could make the present study more comprehensive and substantive.

Chapter VI is followed by a list of reference material used in this study. Lastly, the graphics simulation source code is attached as appendices.

II. REVIEW OF PREVIOUS WORK

A. INTRODUCTION

Research in autonomous vehicles has been ongoing for many years. Initially, progress was slow and difficult, partially because of the limitations of available technology. The results of early research indicate that many areas of autonomous vehicle research are extremely complex and remain open to study. However, in recent years, many advances and technological breakthroughs have made feasible tasks which were previously impossible.

Some of the research that has been completed on autonomous vehicles is presented in this chapter to illustrate the nature and complexity of the problems encountered. Additionally, this chapter contains a brief discussion on vehicle dynamics. Lastly, the current status and limitations of computer vision in relation to autonomous vehicles are discussed.

B. VEHICLE DYNAMICS

There are many complex interactions occurring between a moving vehicle and its environment which must be considered in any study involving vehicle dynamics. Some of these interactions are discussed here to provide insight on subjects which may have an impact on the research at hand.

1. Tire and Road Forces

The main forces arising between the tires and the road are those resulting from acceleration and deceleration in the forward or reverse direction and from turning. The forces caused by acceleration and braking of a vehicle are related to the load carried by the wheels and the coefficient of friction. Cornering a vehicle produces complex lateral forces, making computations difficult, especially when acceleration or braking is involved. Nearly all tire wear is the result of these forces. [Ref. 6]

2. Rolling Resistance

Rolling resistance exists because tires are not rigid and change shape as they come in contact with the roadway. When the wheel turns, the portion of the tire coming in contact with the road bulges and flexes. As the rotation continues and this portion of the tire leaves the road, the tire returns to its original shape. Because the tire momentarily changes shape, energy is consumed. The resistance to this motion is called *rolling resistance.* [Ref. 6]

3. Air Resistance

The amount of resistance attributed to aerodynamic drag depends on many factors such as vehicle shape, frontal size, and velocity. Large, thick vehicle bodies show a predominance of drag due to pressure built up to the vehicle's front as it travels in the forward direction at high velocities. This effect can be minimized in slender streamlined vehicle bodies with smooth contours. [Ref. 7]

4. Slip Angle and Side Forces.

Slip angle and side forces are encountered when a rolling wheel is acted on by a side force. The side force is normally caused by centrifugal forces when the vehicle negotiates a curve. The road usually acts on the tire in a direction opposing the side force. The angular difference between the direction of motion and the true wheel rolling direction is called the *slip angle*. [Ref. 6]

5. Other Effects.

The tires and the vehicle are acted on by other forces too numerous to discuss in this work. Some of these forces include self-aligning torque, gyroscopic effects, roll forces. suspension effects, vibrations, and engine torque.

C. AUTOMATIC SYSTEMS FOR LATERAL AND LONGITUDINAL CONTROL

1. Highway Vehicles.

Since the 1960's, numerous research groups have investigated highway automation as a possible solution to some of the problems posed by an ever-increasing number of motor vehicles [Refs. 8-16]. These traffic problems cannot be solved by building larger and faster highways since the cost of construction is prohibitive. Additionally, as vehicle speeds increase, highway safety becomes an important issue. In the late 1960's, Fenton and Olson suggested a solution involving automated highways [Ref. 8]. On the automated highway, traffic flow is increased by decreasing the distance between vehicles through automatic control. The authors hoped that such a system could greatly increase lane capacity at high

speeds while still reducing the number of highway accidents. In this concept. described in [Ref. 8], each vehicle has two modes; a manual mode for use on rural roads where the vehicle is controlled by the driver. and an automatic mode for use on the automated highways.

To control vehicle speed and spacing, some type of longitudinal controller is required. This could be a centralized computing system which is capable of maintaining a complete overview of the traffic at all times or an independent system installed on each vehicle. Additionally, each vehicle requires sensor equipment to measure vehicular spacing and relative velocity. The information provided from such measurements must be analyzed and appropriate control signals sent to the vehicle.[Ref. 8]

In another study of vehicle automatic longitudinal control, Bender, Fenton, and Olson investigated a car following system to increase both highway capacity and highway safety. The research vehicles in this work were instrumented so that braking, acceleration, and steering are managed by an automatic control system. Road tests were then conducted using several vehicles. The tests were designed to gather data on the acceleration and deceleration of the following vehicle in response to velocity changes by the lead vehicle. [Ref. 9]

The results of the various road tests show that automatic system performance can be superior to that of a human driver, especially in situations where emergency braking is required. Of note is that reaction time to full braking was 0.4 seconds for the automated system compared to 0.5 - 1.0 seconds for an

alert driver in a similar situation. In many cases, if the following vehicle had been under driver control when the lead vehicle applied emergency braking, a rear end collision would probably have resulted. [Ref. 9]

In later work, Fenton and Chu focused on control of vehicles entering the highway environment and on control of vehicles traveling on the highway. Here, a vehicle controller was designed and tested under full scale conditions. The vehicle was capable of responding to all non-emergency commands in a satisfactory manner while maintaining passenger comfort. [Ref. 13]

Additionally, other research studies have been conducted on lateral steering control. Fenton, Melocik, and Olson used a vehicle which tracked a cable buried beneath the surface of the roadway. Sensors on the vehicle measured the magnetic field produced by the current in the wire. Then, the measurements were processed to provide the vehicle's position with respect to its driving lane. Several different controllers were designed and tested under full scale conditions. Results of the tests indicate that excellent lateral control, good insensitivity to disturbances, and comfortable ride could be obtained using a simple single loop controller. [Ref. 11]

The term *mechatronics* is applied when electronic components have been intimately integrated with mechanical systems. In automobiles, this technique provides designers with new opportunities to improve automobile handling, performance, and safety. El-Deen and Seireg use mechatronics to improve a vehicle's ability to perform, even when traveling at high speeds on low friction

surfaces [Ref. 17]. Their concept uses a miniaturized computer system incorporated in the vehicle's power steering unit. The computer reads preprogrammed rules to enhance the vehicle's steering performance during critical situations when a driver may not have sufficient time or experience to react.

With a computer simulation, the authors were able to show that the proposed system improved vehicle stability in critical road conditions. The vehicle's computer-controlled steering system uses the preprogrammed rules to implement corrective steering inputs in real time. This concept could readily be expanded to include control of vehicle braking and acceleration, thus providing improved overall performance. [Ref 17]

2. Automated Guided Vehicles.

The concept of the automated guided vehicle (AGV) system embraces all transportation systems that can function without a driver. The AGV is a flexible unit suitable for simple transport operations with a small number of destinations or for complex and centrally-controlled transport processes [Ref. 18]. A study of AGV's includes all types of driverless industrial trucks, such as fork trucks and electric tractors. Due to industrial demands, AGV's also include different types of conveyor assemblies and trolleys. However, the discussion to follow is limited to driverless industrial trucks.

Successful research with process control by a central computer system and the use of on-board microprocessors have allowed AGV Systems to compete with

other truck systems. Each installation consists of several components which include the trucks, the network, the load handling system, the truck controller, and a traffic controller. The trucks and the load handling system are of the standard industrial type and are not discussed in detail here. However, the reader should understand that each truck is modified for steering control, speed control, and load management by the automatic systems.

The network usually consists of a guidance system and signal devices for information transfer. Networks range from a simple closed course in small factories to complex systems with multiple guidewires and switches providing many possible routes in large installations. Information transfer is simple and most often passive. Signal devices such as permanent magnets, infrared transmitters, or light signals provide information to the on-board processor concerning the current vehicle location, speed restrictions, or other pertinent data, as necessary for a specific implementation.

Most on-board truck controllers have several functions which include:

- lateral steering control to keep the truck on course,
- route control which moves the truck through a network to its goal, and
- longitudinal speed control and braking.

In typical systems, lateral steering is controlled by a track or the magnetic field surrounding an electrically excited guidewire beneath the surface of the road. In the guidewire implementation, two coils on the vehicle are used to

sample the magnetic field produced by the current carrying wire. If the truck deviates from the intended course, a difference in the voltage produced by the two coils is noted and triggers a control signal to initiate a steering correction.

The longitudinal speed control portion of the truck controller receives information from several sources. These sources include network signal devices, vehicle separation devices, and the route controller. Most trucks have only a few operating speeds, making the task of the longitudinal speed controller simpler.

Route planning is the most difficult problem in the control system design and can be accomplished on-board the truck by a processing unit, at a central processing computer, or with a combination of on-board processing and a central computer. Typically, small companies with a limited number of trucks cannot afford the expense of a data link between individual trucks and the central computer. Additionally, an on-board route processing system provides added reliability over a single centralized system.

Regardless of the type of route planning system that is implemented, the control process involves job planning, job management, truck planning, truck management, and optimization. The task of a centralized computer processor is complex and can involve numerous priorities and prerequisites as determined by the network, the task, and management.

With on-board route planning, the vehicle destination is typically input by means of data transmission as the vehicle passes a certain position in the

network. Thus, a certain portion of the planning function is centrally controlled. This technique forms a hierarchical structure and allows the system to be autonomous at all levels.

3. Ohio State University Adaptive Suspension Vehicle

The Ohio State University Adaptive Suspension Vehicle (ASV) is a prototype legged robot intended to provide performance characteristics desirable in a military rough terrain vehicle. The ASV is a hexapod that uses statically stable gaits while moving. It can carry its operator and an internal payload of 500 pounds. The machine is powered by a motorcycle engine that drives a flywheel and 18 variable displacement hydraulic pumps. [Ref. 19]

The ASV is equipped with a network of 17 microcomputers to coordinate foot placement, leg movement, and the control of body attitude, thereby freeing the operator for higher level decisions concerning speed and direction. The operator can control forward and lateral velocity, rate of turn, body height, and attitude, if necessary. [Ref. 19]

To ensure careful foot placement in rough terrain, the ASV is equipped with a sophisticated terrain scanning system that is based on continuous wave phase comparison using infrared light. The light beam is scanned at a 2 Hz rate in a raster pattern covering the ground ahead of the ASV to a range of about 10 meters.

Outdoor trials of the ASV hexapod were initially conducted in 1986. During the first tests, the walking machine demonstrated that it is capable of

movement in open fields at speeds in excess of 2 mph. It can also negotiate simple obstacles and ditches. Tests are continuing.

4. FMC Corporation Autonomous Land Vehicle

The aim of this project is to develop an automatic pilot that can control an autonomous land vehicle in real time. The system is based on the M113A2 tracked and armored personnel carrier that is controlled from a command trailer. [Ref. 20]

This system uses a hierarchical control architecture with subsystems named the Planner, the Observer, the Mapmaker, the Pilot, and the Vehicle Controller. As indicated by its name, each subsystem has a well-defined and important function to perform. [Ref. 20]

The highest control system, the Planner, uses digitized maps to select a general path to the goal for its mission. The Planner is able to incorporate a variety of mission requirements into the path selection process. Typical mission requirements can include minimizing detection of the vehicle by the enemy, minimizing the time of the mission, or minimizing the energy consumption to accomplish the mission. The path provided by the Planner is in an abstract form consisting of segments with left and right boundaries. These segments are output in LISP syntax and include a general heading to the goal and a maximum vehicle velocity for that segment.

The main function of the Observer is situation assessment and resolution. This is based on the segmented path received from the high level Planner and on

information received from on-board sensors such as an obstacle detector and an inertial land navigation system. The output of the Observer is a more usable plan for the next subsystem, the Mapmaker.

The Mapmaker generates the Pilot Map containing the global path border, the nearest obstacle borders, and the sensor visibility limits. Here, the output of the obstacle detection sensor is combined with the plan from the Observer to create the Pilot Map.

The Pilot Map is a local map in a format that the Pilot can use. It is the Pilot's responsibility to guide the vehicle along a feasible route staying within the constraints provided by the Planner and avoiding unforeseen obstacles. The Pilot is able to generate subgoals and select an optimum path in real time because of the hierarchical nature of the system. Once an optimal path is selected, the Pilot issues instructions to the lowest level subsystem, the Vehicle Controller.

Field tests of the FMC Corporation Autonomous Land Vehicle indicate it can perform obstacle avoidance successfully and complete path execution at 5 mph. The system is now being improved to deal with more complex terrain features. [Ref. 20]

5. Martin Marietta Autonomous Land Vehicle

The Martin Marietta Autonomous Land Vehicle is an eight wheel all-terrain vehicle which is mechanically capable of traveling up to 18 mph on rough terrain and up to 45 mph on improved surfaces. This vehicle also uses a hierarchical system to provide autonomous navigation and tactical decision

making. The main modules in this design are the Reasoning Subsystem, the Perception Subsystem, and the Control Subsystem. [Ref. 21]

The Perception Subsystem incorporates various sensors to provide real time input on the surrounding terrain. A three-dimensional model of the local terrain is generated by combining input from the different sensors. This model of the terrain is used by the Reasoning Subsystem for path planning.

The Reasoning Subsystem consists of a goal seeker, a navigator, and a knowledge base. This subsystem interprets the mission goals and any limitations which may apply. The goalseeker analyzes the mission and uses the knowledge base to provide the navigator with numerous subgoals oriented toward the mission goal. The navigator combines the subgoals with the three-dimensional model from the Perception Subsystem to provide several possible trajectories. Finally, cost functions are applied to the possible trajectories to determine one route for the vehicle to follow. The Control Subsystem converts the selected route to steering and speed commands to drive the vehicle. [Ref. 21]

D. COMPUTER VISION

Computer vision is an important subfield of artificial intelligence. A strong demand exists for computer vision applications in nearly every area of robotics. Still, it is a slow, difficult process and there is a natural desire by researchers to understand human vision as a problem which could result in the development of a general methodology for solving computer vision tasks. [Ref. 4:pp. 255-277]

The problem of computer vision is to give a computer a picture of a scene and have the computer determine what the objects are in the scene and what their spatial relationships are [Ref. 22]. The most powerful computers often require more than a minute to process a single scene that a human can interpret in the blink of an eye. As a result, at present, it appears impossible to interpret image sequences in real time for a moving autonomous robot [Ref 4:pp. 301-308].

Another difficulty with computer vision is the combining of sensor input with intelligence to perform vision. As long as the robot or vehicle is moving along a country road with occasional houses and trees, the knowledge of what is expected is available. But if the vehicle is allowed to move into a different environment with different terrain and varied scenery, with current knowledge representation and knowledge learning techniques, it is impossible to give the computer adequate information or time to analyze its new surroundings. [Ref. 22]

Considerable progress has been made in the field of computer vision in the last decade, especially in the areas of industrial machine vision systems that use simple image processing and pattern recognition. However, more work is needed in the area of autonomous vehicles in order to provide consistent real time data.

E. SUMMARY AND CONCLUSIONS

A brief overview of vehicle dynamics is provided in this chapter. Additionally, some of the research in a variety of different autonomous vehicles has been reviewed to gain an understanding of the complexity involved. It is

noted that sophisticated systems such as autonomous land vehicles require specialized high performance parallel processing systems to operate in the environment demanded of them. The ability of an autonomous vehicle to complete its mission also depends on the speed, accuracy, and quality of its computer vision system. Without these requirements, autonomy is very difficult, if not impossible to achieve.

In the next chapter, the problem statement for this work is refined. The assumptions for the selected vehicle model are detailed and the vision model is described. As the chapter proceeds, the control model used for the simulation in this work is developed and linearized.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

In this chapter, the model used for the three-dimensional graphics simulation is described in detail. To assist the reader unfamiliar with control theory, a brief description of the purpose of developing a mathematical model is given. Additionally, a linearization of the mathematical model is included.

The aim of this research is to examine and study longitudinal speed control and braking. This serves as motivation for developing a three-dimensional graphics simulation model. This technique attempts to mimic the way a human controls the speed of his vehicle on the road, with particular attention to the aspect of braking.

The hypothesis is that human drivers make errors estimating the distance to a desired stopping point. We also make errors estimating vehicle speed and estimating vehicle response when the brake is applied. Humans develop an acceleration and deceleration plan, which we unconsciously maintain while we are driving. Through experience and training, humans have an approximate model of vehicle maximum braking. vehicle stopping distance, and vehicle acceleration and deceleration capability. The acceleration and deceleration plan, which this work is primarily concerned with, is some fraction, α , of the vehicle's maximum stopping capability, where $0 < \alpha \leqslant 1$.

The amount by which α varies from the maximum vehicle stopping capability depends on several factors such as the vehicle speed, the available stopping distance, the road surface conditions, the skill and experience of the driver, and the general nature of the environment. Here, the environment includes variables such as traffic conditions, weather conditions, dangerous situations, and the degree of the emergency causing the acceleration and deceleration plan to be executed. In normal stopping conditions $\alpha < 1$, and for emergencies $\alpha = 1$.

B. VEHICLE MODEL

The mathematical model used to describe the vehicle must be simple enough to provide insight into the behavior of the system, yet detailed enough to provide an adequate description of the system. To keep our vehicle model simple, the acceleration and deceleration plan is executed only on straight road sections and the effects of steering on the vehicle are ignored.

Many interactions between the driver, the vehicle, and the environment are also ignored to keep the complexity of the mathematical braking model manageable. It has been assumed that braking is perfect in the sense that the maximum vehicle stopping capability can be reached without skidding, swaying, sliding, or any other unusual braking effects. Additionally, tire dynamics as well as shock and strut effects are ignored. The above restrictions and assumptions eliminate any rotational moments and allow the vehicle to be idealized as a

point mass. The model can be further simplified by allowing velocity and acceleration vectors only along the vehicle centerline, eliminating sideslip angles.

C. ROAD MODEL

This work assumes that near perfect conditions are available for the autonomous vehicle. The only obstacles on the road and in the vehicle path are stop signs and traffic signals with green, yellow, and red lights. The signal devices are called *semaphores.* These seemingly unreasonable assumptions were made to allow the author to concentrate on the deceleration plan rather than on obstacle avoidance or image and vision processing, where there are already numerous research activities [Refs. 23-29]. Additionally, the road and surrounding terrain is flat, eliminating the influence hills have on the vehicle as well as restricting the mathematics to two dimensions.

D. VISION MODEL

The vision model for this study is based on that developed by McGhee, Zyda, and Tan. Their model consists of a set of road points representing the center of a closed-course track. In the automatic driving mode of that system, the vehicle continuously selects road points to its front and uses those selected road points as targets to steer towards. [Ref. 30: p.30]

This model extends that notion and associates with each road point a *maximum safe speed,* $\dot{x}_{max}$. This maximum safe speed is the smaller of two possibilities:

- the speed limit for that stretch of road or

- the maximum safe speed to come to a stop at some known point, say the location of a stop sign or semaphore.

This notion is depicted graphically in Fig 3.1.

As the driver approaches a stop sign, his perception of the distance to the stop sign, d_p, and of the vehicle's velocity, v_p, is typically in error. In this study, it is assumed that the distance the driver perceives to the stop sign is proportional to the actual distance to the stop sign. That is,

$$d_p = k_d d_a, \tag{3.1}$$

where d_a is the actual distance to the stop sign and k_d is a distance multiplier. For this study. it is somewhat arbitrarily assumed that $0.8 < k_d < 1.2$ for most drivers. Thus, as the vehicle intersects the driver's acceleration and deceleration plan, the driver makes an initial error estimating the distance to the stop sign. It is further hypothesized here that the driver maintains this error with small deviations as he continues to the stop sign. To reflect this, k_d is modeled as

$$k_d = k_i + k_\epsilon(t), \tag{3.2}$$

where k_i is the estimate the driver makes at the start of the deceleration plan and $k_\epsilon(t)$ is the random error he makes while continuing his deceleration plan. By substituting Eq. (3.2) into Eq. (3.1), d_p, the driver's perceived distance to the stop sign becomes

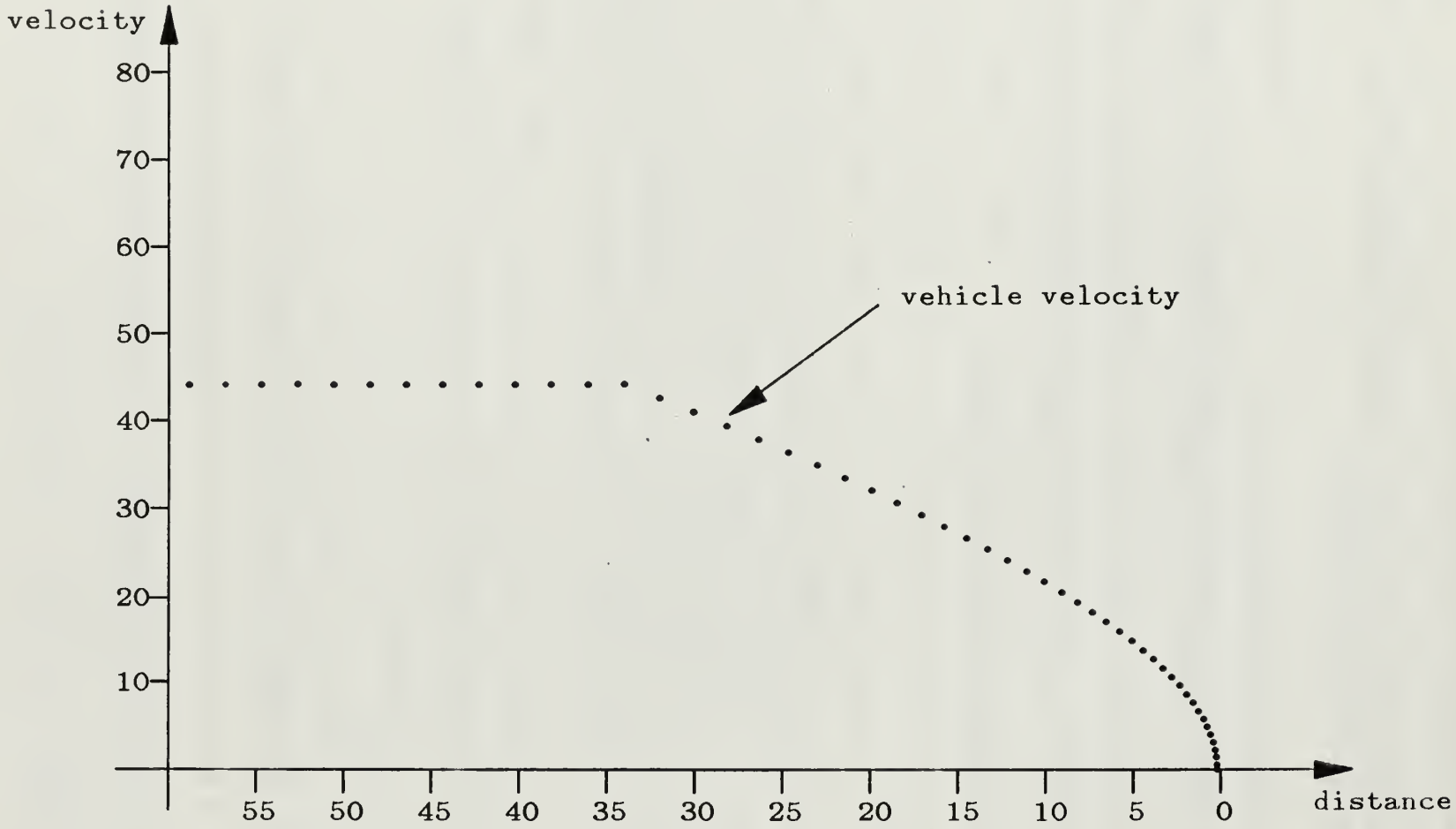


Figure 3.1 Vehicle Approaching a Stopsign.

$$d_p = (k_i + k_\epsilon(t))d_a. \tag{3.3}$$

The driver also makes errors in his velocity estimate, v_p. These errors can be modeled in a similar manner so that

$$v_p = k_s v_a, \tag{3.4}$$

where v_a is the vehicle's actual velocity and k_s is a speed multiplier with typical values assumed to be in the range $0.8 < k_s < 1.2$. The driver also makes an initial error estimating his velocity and maintains this error with small deviations as he continues to approach the stop sign. The speed multiplier k_s can thus be modeled as

$$k_s = k_f + k_n(t), \tag{3.5}$$

where k_f is the estimate the driver makes at the start of his deceleration plan and k_n is the random error he makes while decelerating. Substituting Eq. (3.5) into Eq. (3.4) yields

$$v_p = (k_f + k_n(t))v_a. \tag{3.6}$$

The magnitude of the errors the driver makes in his estimates d_p or v_p depends on many influencing factors such as attentiveness, driver skill, road and weather conditions, etc. However, a study of all of these factors is beyond the scope of this work. Rather, in what follows, all of the multiplying factors appearing in Eq. (3.1) through Eq. (3.6) are simply treated as independent Gaussian random processes denoted $N(\mu, \sigma)$ where μ is the mean of the process

and σ is its standard deviation. That is, at the beginning of any given trial of simulated human braking, a random number generator is used to generate k_i and k_f. Symbolically, from the assumption on the range of k_d and k_s,

$$k_i = N(1, \sigma_i) \tag{3.7}$$

and

$$k_f = N(1, \sigma_f). \tag{3.8}$$

In this formulation, all of the factors involving average driver errors in the perception of speed and distance are combined to determine σ_i and σ_f, with small values representing accurate average perception and large errors inaccurate perception. Since both k_i and k_f have been postulated to typically lie in the interval $[0.8, 1.2]$, a value for σ around $\sigma = 0.1$ is appropriate. That is, if this value is used, average perceived position and velocity are within 20 percent of the true value in 95 percent of the simulation trials.

Since k_ϵ and k_n represent fluctuations in k_d and k_s, it follows that

$$k_\epsilon = N(0, \sigma_\epsilon) \tag{3.9}$$

and

$$k_n = N(0, \sigma_n). \tag{3.10}$$

It is assumed in this study that these fluctuations are small in comparison to the average error. Thus, a typical value for σ_ϵ and σ_n might be in the range $0.01 \leqslant \sigma \leqslant 0.02$.

In the absence of any theory to guide the selection of standard deviations for the random processes discussed above, several representative values are chosen for comparison to real human driving of the vehicle simulation. A more exhaustive investigation of this issue is left to future research.

E. CONTROL MODEL

To develop the extended vision model, assumptions and restrictions stated earlier in this chapter are used. These constraints limit the vehicle to flat straight roads and allow the author to treat the vehicle as a *point mass.* Therefore, for vehicles traveling down the highway, the shortest distance d to a stop sign at which braking can commence is

$$d = \frac{1}{2} a_{\max} \tau^2 \tag{3.11}$$

where τ is the *time to go* to the stop sign defined as

$$\tau = t_{stop\ sign} - t \tag{3.12}$$

and $a_{\max}$ is the maximum braking acceleration. In this equation, t represents the clock time at the current vehicle position while $t_{stop\ sign}$ is the clock time at which the vehicle arrives at the stop sign. Thus,

$$\frac{d\tau}{dt} = -1. \tag{3.13}$$

From Eq. (3.11), the maximum safe speed defined in the vision model is

$$\dot{x}_{max} = a_{max}\tau. \tag{3.14}$$

The minimum value for τ can be determined from Eq. (3.11) and substituted into Eq. (3.14), providing a maximum safe speed $\dot{x}$ at any distance d from the stopping point. The result is a maximum braking curve determined by

$$\dot{x}_{max} = \sqrt{2da_{max}}. \tag{3.15}$$

A graph of the maximum braking curve is shown in Fig 3.2.

For a driver desiring to stop his vehicle at a stop sign, the brake must be applied *prior* to reaching the last possible braking point. This can be accomplished by employing the previously discussed acceleration and deceleration plan in which, during braking, the vehicle acceleration is

$$a_{brake} = \alpha a_{max}, \quad 0 < \alpha \leqslant 1. \tag{3.16}$$

If $\alpha = 1$, the driver has used maximum braking and the vehicle velocity is shown by Fig 3.2. For routine braking situations, $0 < \alpha < 1$ and a typical graph is as shown in Fig 3.3.

F. LINEARIZED ANALYSIS

For a vehicle, the braking deceleration is a function of brake pedal depression δ such that

$$a_{brake} = k_{brake}\delta, \quad k_{brake} > 0. \tag{3.17}$$

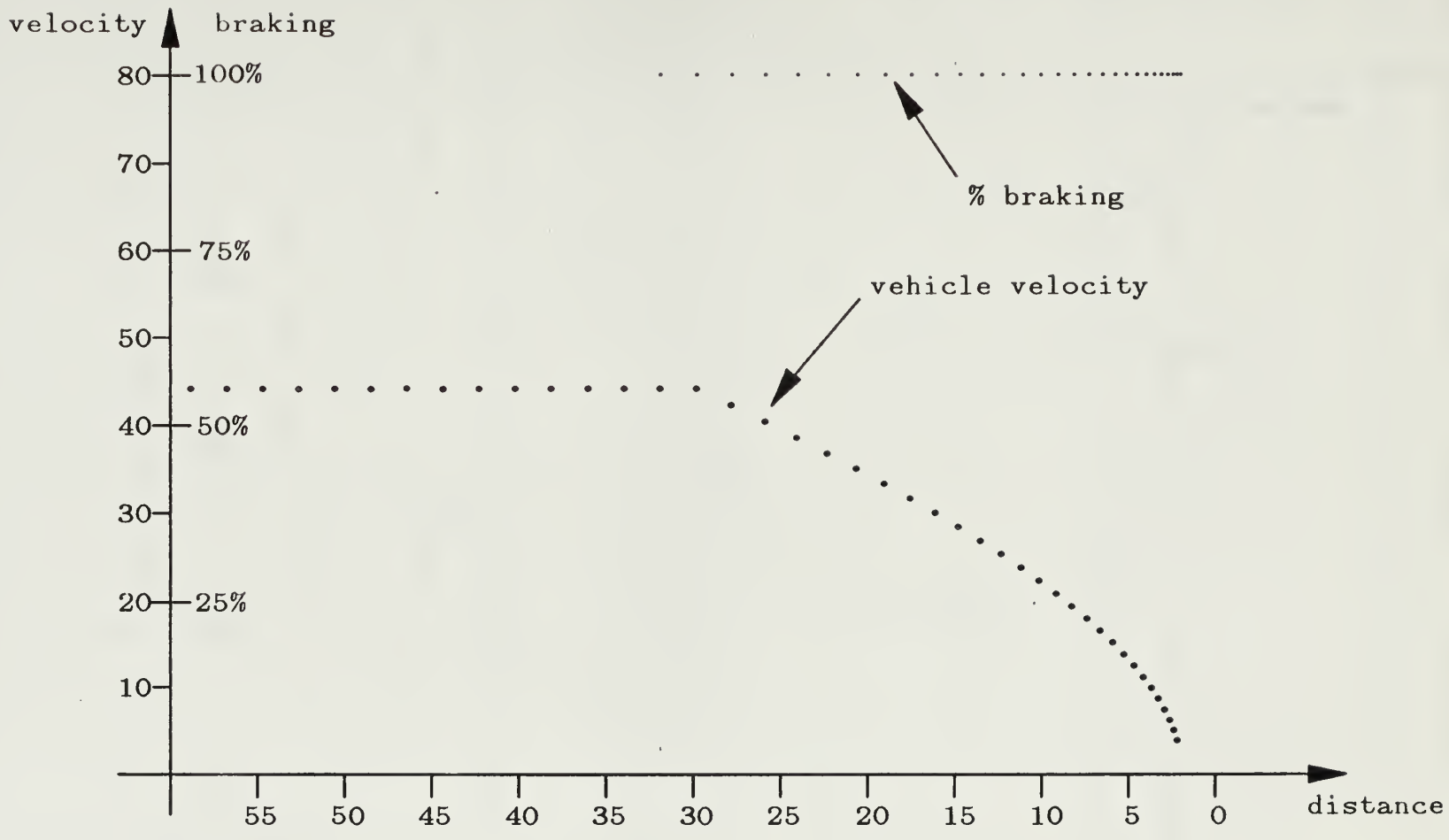


Figure 3.2 Vehicle Velocity with Maximum Braking.

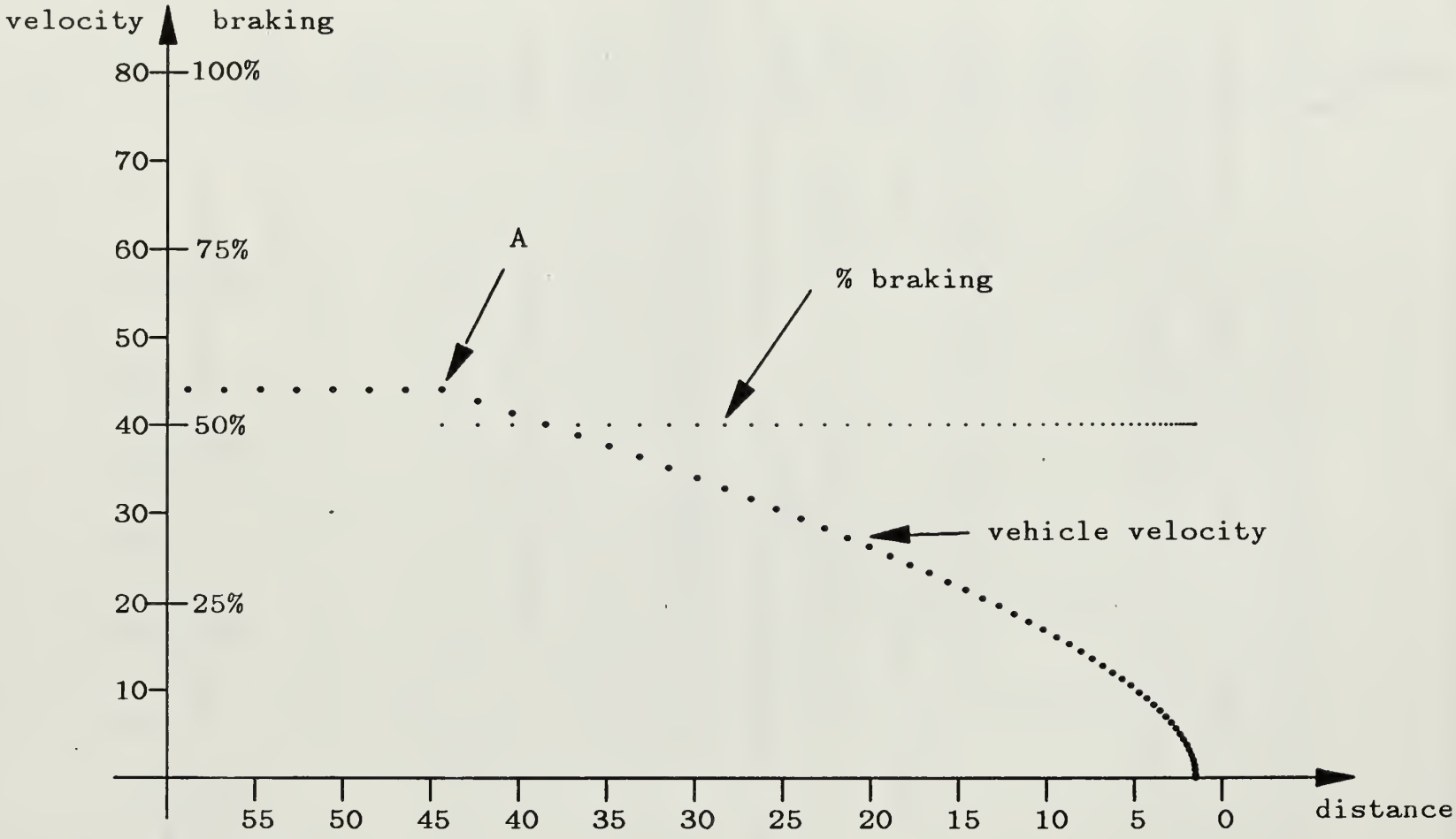


Figure 3.3 Vehicle Velocity with Routine Braking.

Additionally, the engine provides some braking. A representation of engine braking was developed by McGhee, Zyda, and Tan [Ref. 30: p.33] as

$$a_{engine} = \frac{1}{\tau_a} (\dot{x}_c - \dot{x}) \tag{3.18}$$

where $\dot{x}_c$ is the commanded velocity resulting from accelerator depression and τ_a is the acceleration time constant. Now the total acceleration due to braking is

$$\ddot{x} = - a_{brake} + a_{engine}. \tag{3.19}$$

After substitution of Eq. (3.18), this becomes

$$\ddot{x} = - a_{brake} + \frac{1}{\tau_a} (\dot{x}_c - \dot{x}). \tag{3.20}$$

Substituting Eq. (3.17) into Eq. (3.16) yields a brake pedal depression value of .

$$\delta = \frac{\alpha}{k_{brake}} a_{\max} - \frac{1}{\tau_a \, k_{brake}} \dot{x} \tag{3.21}$$

where it is assumed that the driver removes his foot from the accelerator before applying the brake. This value is to be applied at the moment the vehicle intersects the deceleration plan, shown as point A in Fig 3.3. However, drivers are human and often make errors judging the distance to the stop sign and estimating the vehicle velocity. To remedy these problems, two integral terms are added to Eq. (3.21), resulting in the brake pedal depression equation

$$\delta = + \frac{\alpha}{k_{brake}} a_{max} + k_p(d_{plan} - d_p) - k_v(v_{plan} - v_p)$$

$$- \frac{\dot{x}}{\tau_a k_{brake}} \tag{3.22}$$

where d_{plan} is distance from the stop sign in the driver's acceleration and deceleration plan, v_{plan} is the closing velocity, d_p is the distance to the stop sign as perceived by the driver, v_p is the velocity of the vehicle as perceived by the driver, k_p is the position gain factor, and k_v is the velocity gain factor.

By substituting Eq. (3.22) into Eq. (3.17) and using the result in Eq. (3.20), the vehicle net acceleration is given by

$$\ddot{x} = - k_{brake} k_p(d_{plan} - d_p) + k_{brake} k_v(v_{plan} - v_p)$$

$$- \alpha a_{max} + \frac{1}{\tau_a} \dot{x}. \tag{3.23}$$

Since the vision model assumes that the *average* driver error in estimating distance and velocity is zero, for purposes of linearization, from Eq. (3.1) it can be assumed that

$$d_p = d_a. \tag{3.24}$$

For the same reasons from Eq. (3.4),

$$v_p = v_a = \dot{x}. \tag{3.25}$$

To continue the linearization, it must be noted that if x is taken as the distance

from the vehicle starting point to its current location, and x_{stop} is the location of the stop sign, then

$$d_p = d_a = x_{stop} - x \tag{3.26}$$

and thus

$$d_{plan} - d_p = d_{plan} - (x_{stop} - x). \tag{3.27}$$

Using Eq. (3.25) and this result, Eq. (3.23) becomes

$$\ddot{x} = - k_{brake} k_p (d_{plan} - x_{stop} + x) + k_{brake} k_v (v_{plan} - \dot{x}) - \alpha a_{\max} \tag{3.28}$$

or

$$\ddot{x} + k_{brake} k_v \dot{x} + k_{brake} k_p x = k_{brake} k_p (x_{stop} - d_{plan}) + k_{brake} k_v v_{plan} - \alpha a_{\max}. \tag{3.29}$$

Using standard control theory techniques, the characteristic equation associated with Eq. (3.29) is

$$\lambda^2 + k_1 \lambda + k_0 = 0 \tag{3.30}$$

where

$$k_1 = + k_v k_{brake} \tag{3.31}$$

and

$$k_0 = + k_p k_{brake}. \tag{3.32}$$

From Eq. (3.26), the associated eigenvalues are

$$\lambda = -\frac{k_1}{2} \pm \left[\left(\frac{k_1}{2}\right)^2 - k_0\right]^{\frac{1}{2}}. \tag{3.33}$$

For critical damping, eigenvalues λ_1 and λ_2 must be equal, real, and negative. [Ref. 31] This occurs if the second term of Eq. (3.33) is zero, giving

$$k_0 = \frac{k_1^2}{4} \tag{3.34}$$

and

$$\lambda = -\frac{k_1}{2}. \tag{3.35}$$

G. SUMMARY

The basis for any computer simulation is a detailed mathematical model. The purpose of this chapter has been to develop a model which can be used in a computer simulation to study longitudinal speed control. In the next chapter, the programming environment and the computer simulation model are discussed in detail. A user's guide is also provided.

IV. SIMULATION MODEL

A. INTRODUCTION

Several methods for implementing the mathematical model developed in the previous chapter were examined during the formulation of this study. The design selected by the authors is an extension of the three-dimensional color graphics animation model implemented by McGhee, Zyda, and Tan in work closely paralleling this research [Ref. 30:pp. 46-67].

To support this work, the simulation implemented by McGhee, Zyda, and Tan was modified to provide vehicle control through manual methods or with an autosteer/cruise control system. This was facilitated by networking two IRIS (Integrated Raster Imaging System) workstations, one with the vehicle simulation and one with a vehicle controller. The two IRIS workstations communicate over the Local Area Ethernet Network in use at the Naval Postgraduate School. The selected design provides several different control modes for vehicle operation and enables a user to control the simulation from either IRIS workstation with manual methods, autosteer/cruise control, or a combination of manual, autosteer, and cruise control.

The remainder of this chapter discusses the programming environment, the three-dimensional graphics simulation, and the networking techniques used. First, the programming environment is discussed. Next, the display on each

workstation is described in detail. Also, the Local Area Network and its use in this research are examined. Additionally, the autosteer/cruise control system and vehicle operating modes are reviewed. A complete description of all modules and supporting files used for the simulation is provided. Lastly, a user's guide is included.

B. PROGRAMMING ENVIRONMENT

1. Hardware

The IRIS 2400 Graphics Workstation, manufactured by Silicon Graphics, Inc., is the target hardware for the design, development, and implementation of the mathematical model developed in Chapter III of this work. The workstation is a high resolution system capable of combining real-time color graphics while operating under the UNIX operating system [Ref. 32-33].

2. Programming Language

The UNIX operating system on the IRIS workstation supports C, FORTRAN, Pascal, and Franz LISP. The C programming language was chosen for this work since a large portion of the software was produced in C during earlier work by McGhee, Zyda, and Tan [Ref. 30]. Additionally, communications packages developed at the Naval Postgraduate School are implemented in C and require low level system calls which are most easily handled by programs also written in the C language [Ref. 34].

C. DISPLAYS AND DRIVING COURSE

1. Vehicle Simulation on Driver's Display

The vehicle display includes a dashboard with operating instructions and instrumentation on the lower portion of the IRIS monitor and an "out-of-the-windshield" view of the highway environment on the upper portion of the IRIS monitor, as shown in Fig. 4.1. This display has been named the "driver's display" and corresponds very closely to an environment with an *onboard* operator, whereas the display on the second IRIS workstation, shown in Fig. 4.2, has been named the "navigator's display," corresponding to a *remote* control environment. Of course, the driver's display could also be interpreted as part of the remote environment if the vehicle is assumed to be equipped with an on-board television camera and a video data link.

The highway environment consists of a closed-course track constructed with four straight sections of simulated asphalt and four curved sections of simulated asphalt. Intersections, stop signs, speed limit signs, and a semaphore have been added to the software developed by McGhee, Zyda, and Tan [Ref. 30:pp 46-67]. Fig. 4.2 is an overhead view of the closed-course track showing the locations of the semaphore and the stop signs.

2. Control Simulation on Navigator's Display

The navigator's display is executed on an IRIS workstation which is networked to the driver's display. This IRIS system projects an overhead view of the closed-course track on the left portion of the monitor and essential vehicle

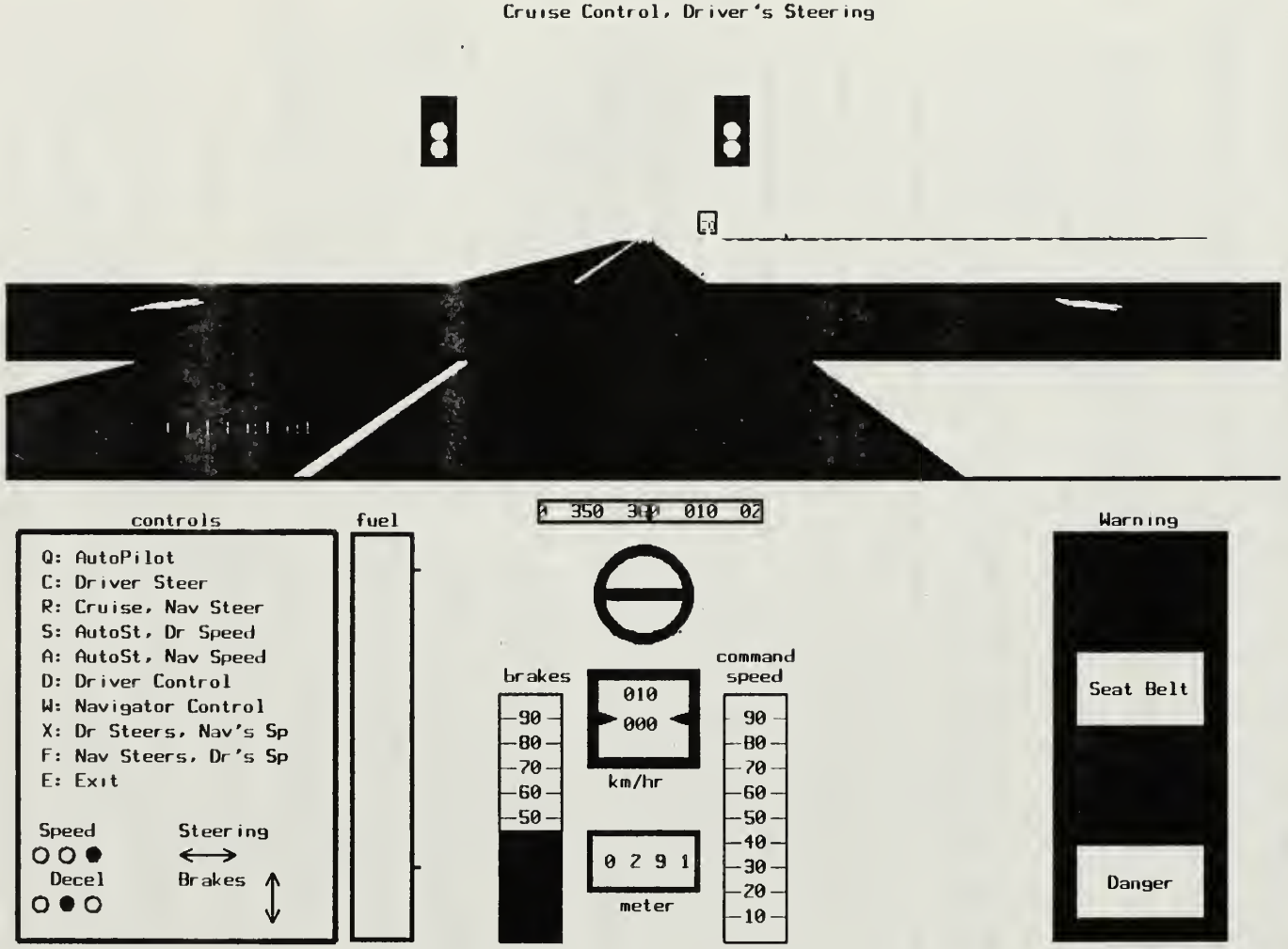


Figure 4.1 An "Out-of-the-Windshield" View on the Driver's Display.

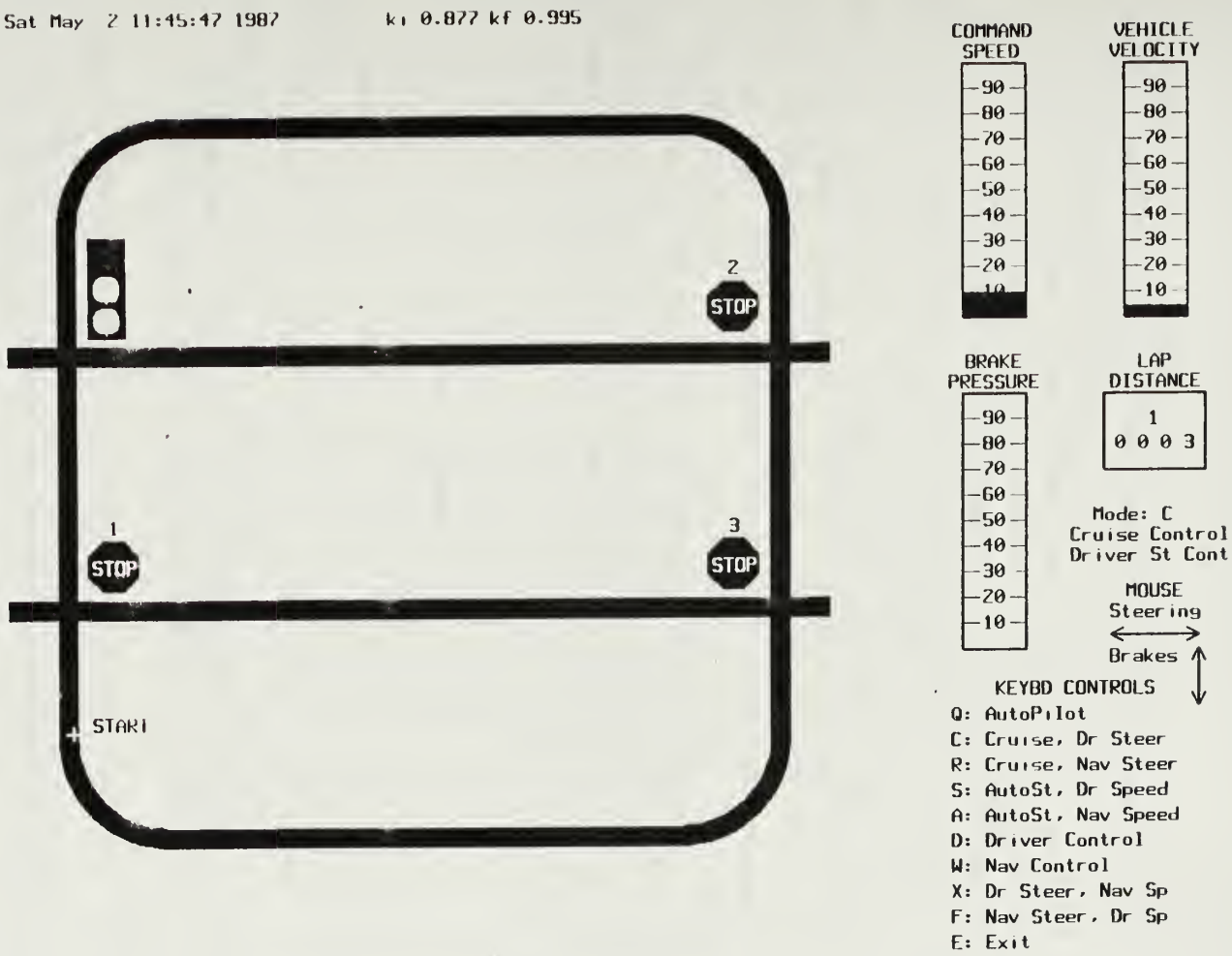


Figure 4.2 A View of the Highway Circuit on the Navigator's Display.

instrumentation on the right portion of the monitor as depicted in Fig. 4.2. Additionally, the progress of the vehicle on the highway is provided by *crosshairs* which are updated as the vehicle moves around the highway circuit.

D. NETWORKING

The two IRIS workstations used in this research communicate on a Local Area Ethernet Network. The Ethernet can transfer serial packets of data at a maximum transfer rate of 10 megabits per second [Ref. 35: pp. 80-86].

In addition to Ethernet, this work utilizes a prototype multimedia computer conferencing system designed by Manley to effect continuous communications between two programs executing on separate computers [Ref. 34]. During program execution, pertinent vehicle information such as vehicle coordinates, distance from the start of the highway circuit, vehicle velocity and vehicle braking information is continually transmitted from the driver's display to the navigator's display. Additionally, the status of the semaphore and the current operating mode are transmitted to the navigator's display. In return, the driver's display receives command and control information from the navigator's display which includes commanded velocity, vehicle braking control, vehicle steering control, and operating mode.

E. DRIVING MODES

The 3D color graphics simulation developed for the mathematical model presented in this work can be controlled by the user with the IRIS workstation's mouse, or by the autosteer/cruise control feature, or by a combination of mouse and autosteer/cruise control.

1. Lateral Steering Control

Two methods of lateral steering control were developed by McGhee, Zyda, and Tan [Ref. 30:pp. 30-45]. Both methods are used in this simulation. The first method allows a user at the driver's display to manually steer the vehicle by observing the "out-of-the-windshield" view on the driver's display and using lateral movement of the mouse on the IRIS workstation. This closely models the "sight picture" and actions a driver would experience behind the steering wheel of an actual automobile, a situation with an *onboard* operator. Since the mouse on the navigator's display can also control the lateral movement of the vehicle model, a *remote* control situation can also be simulated with the user at the navigator's display. Therefore, by selection of the proper driving mode, the user can manually control lateral steering from either IRIS workstation.

The second means by which steering can be controlled is the autosteer method, where the vehicle steers toward target points in the center of its driving lane. As the vehicle proceeds, new targets are continually selected such that the vehicle constantly has a goal to its front.

2. Manual Longitudinal Speed Control

Two methods of longitudinal speed control have been developed in this simulation. In the first, the user can manually control the vehicle velocity and braking by utilizing the mouse on either IRIS workstation and by observing the "out-of-the-windshield" view on the driver's workstation, thus providing a capability to simulate *onboard* or *remote* control of vehicle speed. Acceleration and deceleration are accomplished by clicking the associated IRIS workstation mouse buttons. This simulates the depression and release of the accelerator in an actual vehicle. Braking is manually controlled by vertical displacement of the mouse to apply brake pressure and by return of the mouse toward its original position to reduce or remove the brake application.

Manual longitudinal speed control can be monitored using the instruments provided on the dashboard of the driver's display or with the instruments provided on the navigator's display.

3. Cruise Control

The second method of longitudinal speed control employs an idealized *cruise control* system. This system uses output from a simulated vision system to respond to speed limit signs, stop signs, and a semaphore. In routine circumstances, *cruise control* operates as in a conventional vehicle and maintains a commanded velocity for the driver. However, the *cruise control* system in this work can also set a new commanded velocity based on speed limit signs observed by the simulated vision system as the vehicle travels the closed-course track.

Additionally, the *cruise control* system can react to a stop sign or semaphore and guide the vehicle to a smooth stop. After an appropriate delay at a stop sign, or when the semaphore turns green, the *cruise control* system accelerates the vehicle through the intersection while the simulated vision system watches for new signs.

To maintain longitudinal speed control, the *cruise control* module analyses the vehicle location, the vehicle velocity, and semaphore/stop sign information using the mathematical model developed in Chapter III. Based on the results of this analysis, control information in the form of commanded velocity (accelerator position) and brake position is transmitted from *cruise control* on the navigator's display to the vehicle on the driver's display.

The *cruise control* method of longitudinal speed control can be monitored with the instruments provided on the dashboard of the driver's display or with the instruments on the navigator's display.

4. Modes of Operation

The simulation developed in this research is designed to be controlled from either the driver's display (an *onboard* control situation) or from the navigator's display (a *remote* control situation). Additionally, the user can select any combination of the lateral and longitudinal control methods previously described. Thus, the user has nine different driving modes as detailed in Fig. 4.3. The user can change driving modes at any time with a single keystroke at the keyboard of either display.

Keyboard Input	Mode	Description
D	DrManual	Driver controls steering and speed.
W	NavManual	Navigator controls steering and speed.
C	CruiseDrSteer	Cruise control speed, driver controls steering.
R	CruiseNavSteer	Cruise control speed, navigator controls steering.
S	ASteerDrSpeed	Automatic steering, driver controls speed.
A	ASteerNavSpeed	Automatic steering, navigator controls speed.
Q	AutoPilot	Automatic steering and cruise control.
X	DrSteerNavSp	Driver controls steering, navigator controls speed.
F	NavSteerDrSp	Navigator controls steering, driver controls speed.
E	Exit	Terminates Simulation.

Figure 4.3 Operating Modes and Terminal Input.

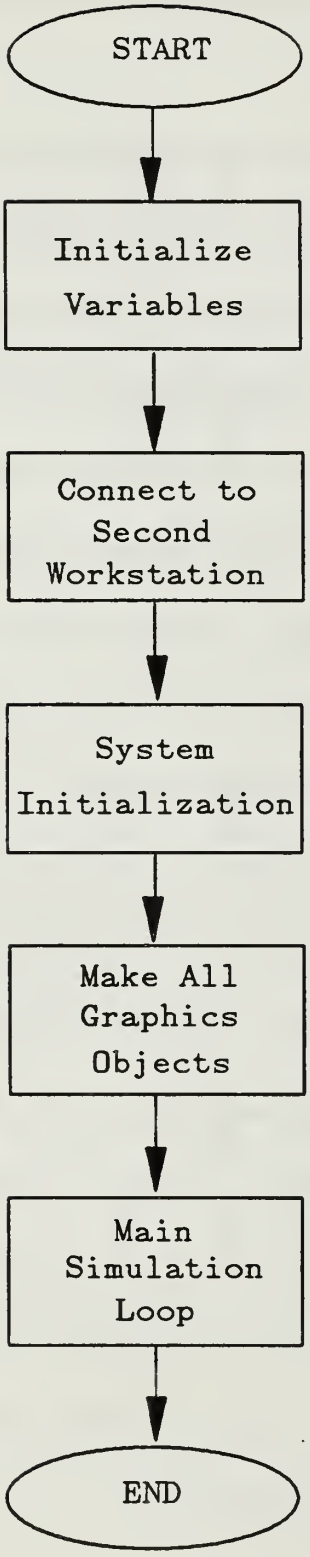
F. MODULE DESCRIPTION ON THE NAVIGATOR'S DISPLAY

1. Navigate.c

This module is the control module for the navigator's graphics simulation. It initializes the IRIS workstation, sets all the local and global variables, and opens a connection to the driver's display on the second workstation. Navigate.c receives and displays the commanded velocity, the vehicle velocity, the vehicle's brake position. the vehicle's distance from the start of the course and the vehicle's coordinate location. Additionally, this module provides command and control information to the driver's display. Lastly, while in manual steering modes, navigate.c handles input received from the mouse on the navigator's display. Fig. 4.4 through Fig. 4.8 is a flowchart of this module.

2. Cruise.c

This module is a key module in the navigator's display and is instantiated only in the cruise control modes. Here, the module calculates the driver's acceleration and deceleration plan using Eq. (3.15). Next, simple vision input is processed with the current vehicle velocity and vehicle location to regulate longitudinal speed control as shown in Fig. 4.9, a flowchart of cruise.c. If the vision input indicates a stop sign within the driver's acceleration and deceleration plan, a call is made to brake.c to stop the vehicle. Brake.c is used in a similar manner during semaphore color processing.



(See Figure 4.5 for more detail)

Figure 4.4 NAVIGATOR.C Flowchart

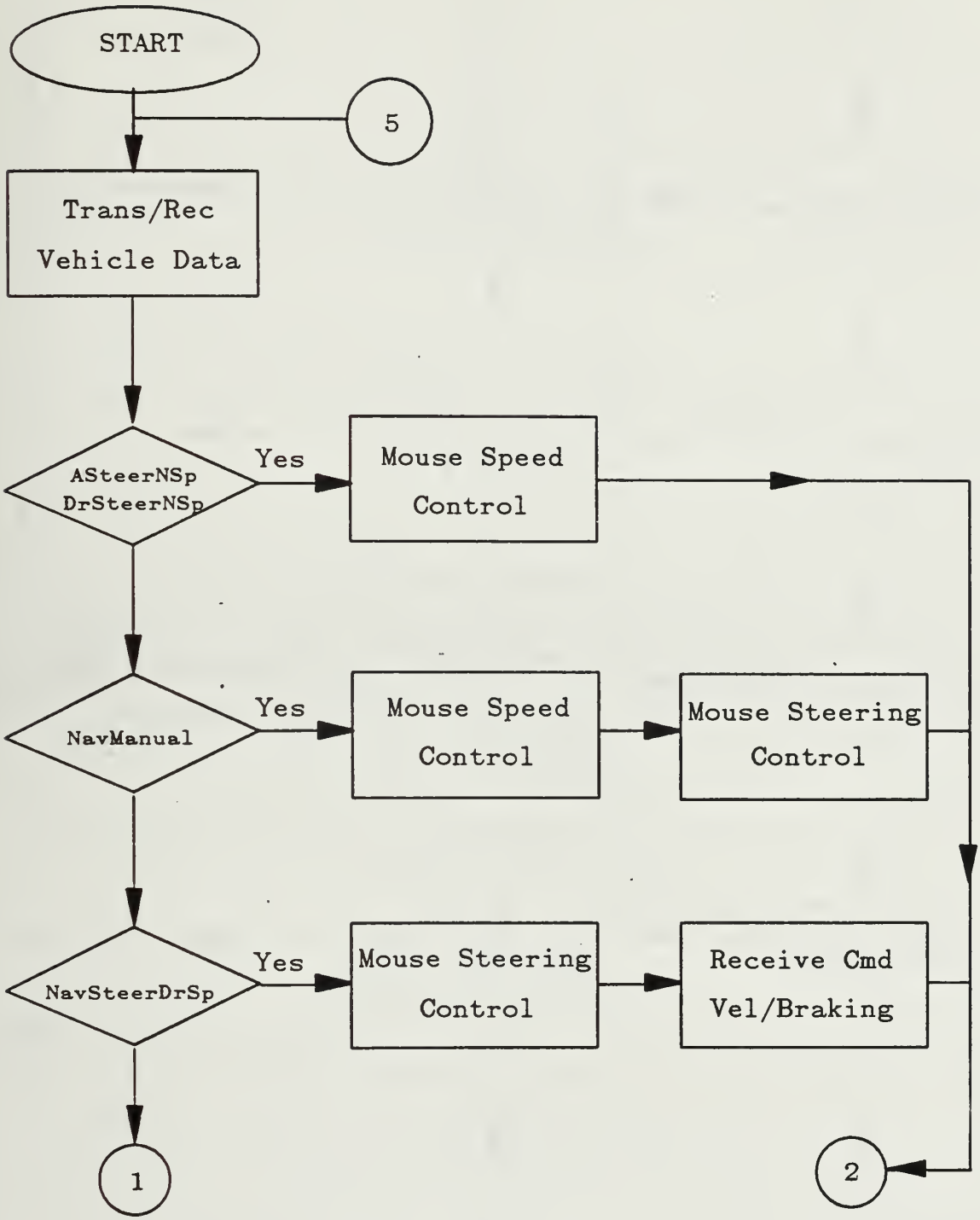


Figure 4.5 Main Loop of Navigator.c (Part 1)

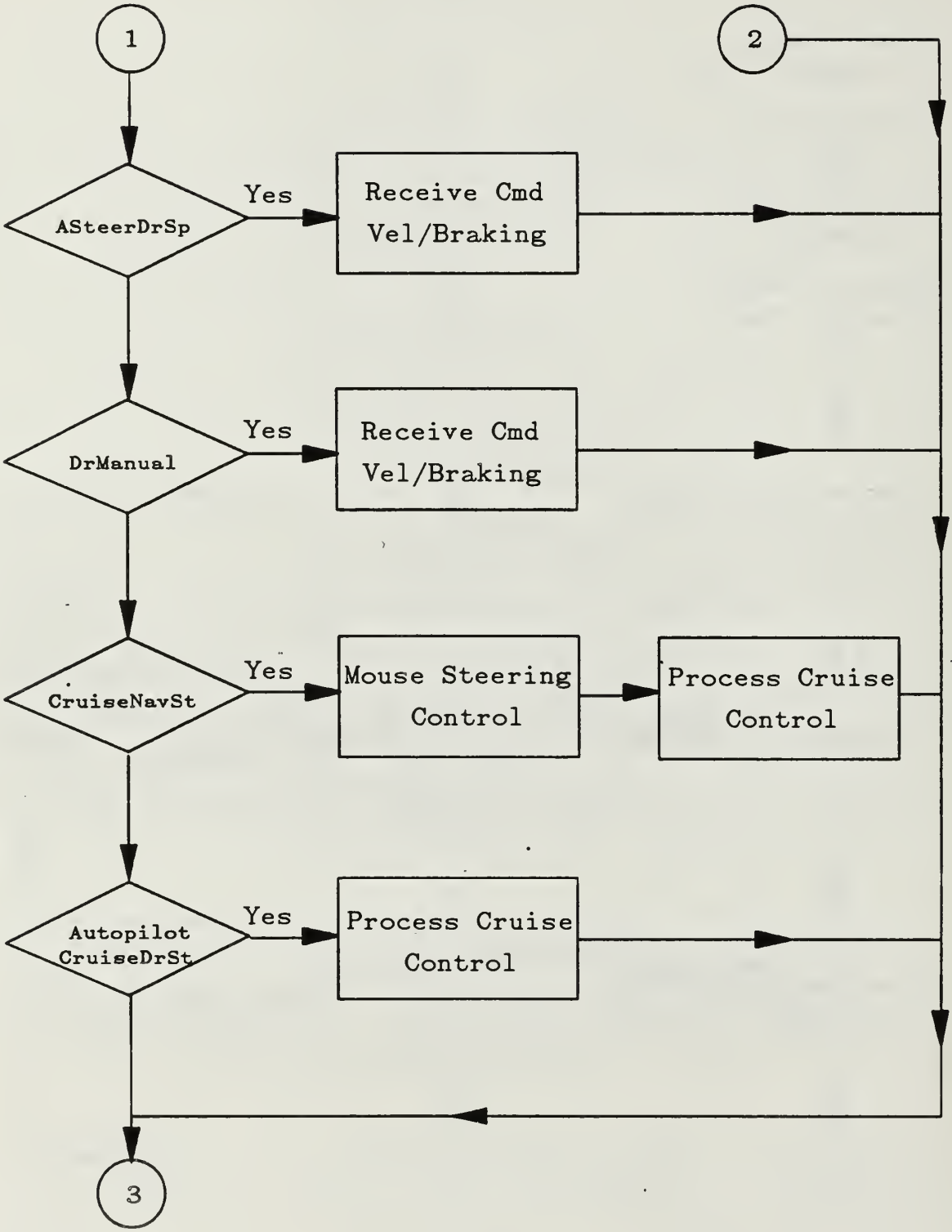


Figure 4.6 Main Loop of Navigator.c (Part 2)

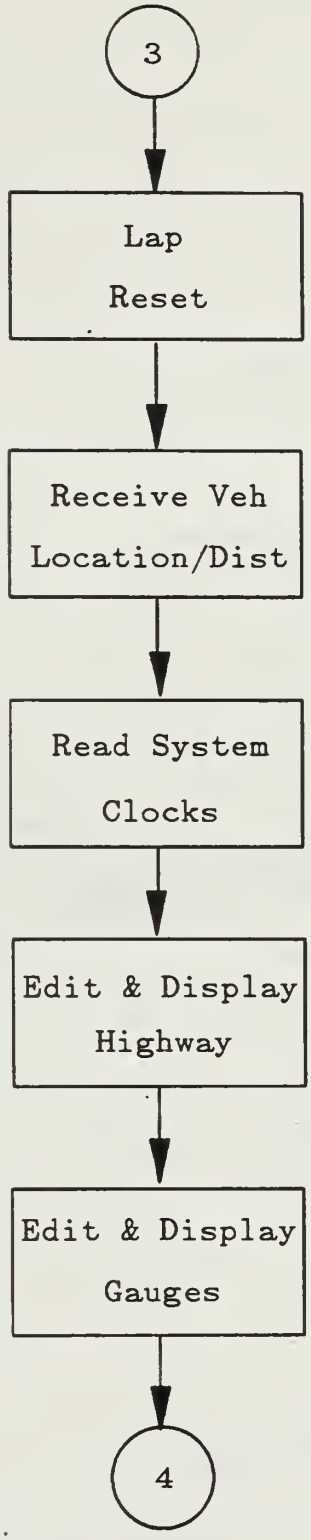


Figure 4.7 Main Loop of Navigator.c (Part 3)

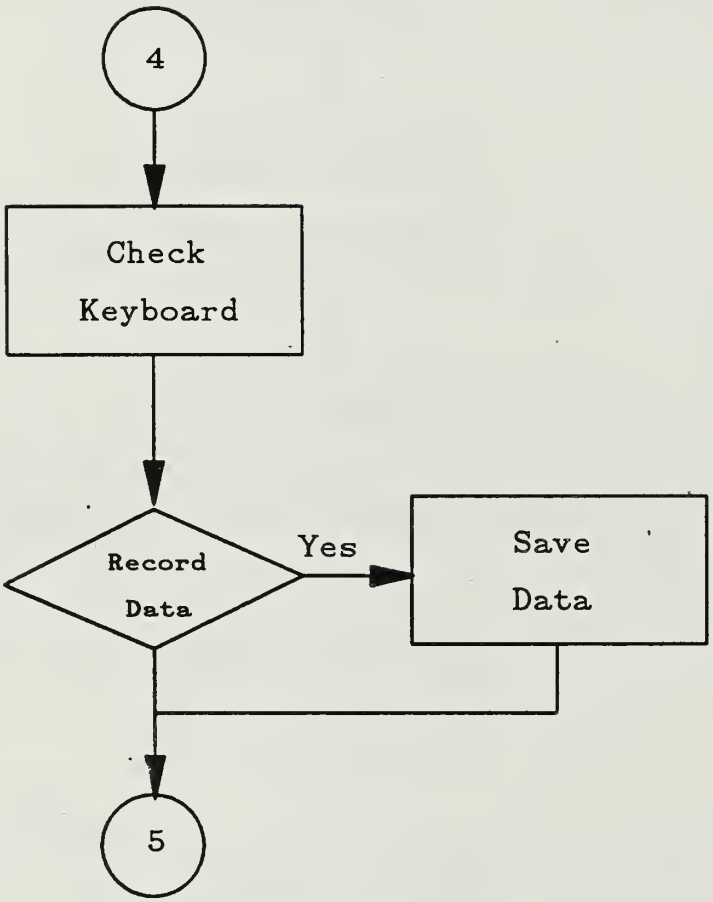


Figure 4.8 Main Loop of Navigator.c (Part 4)

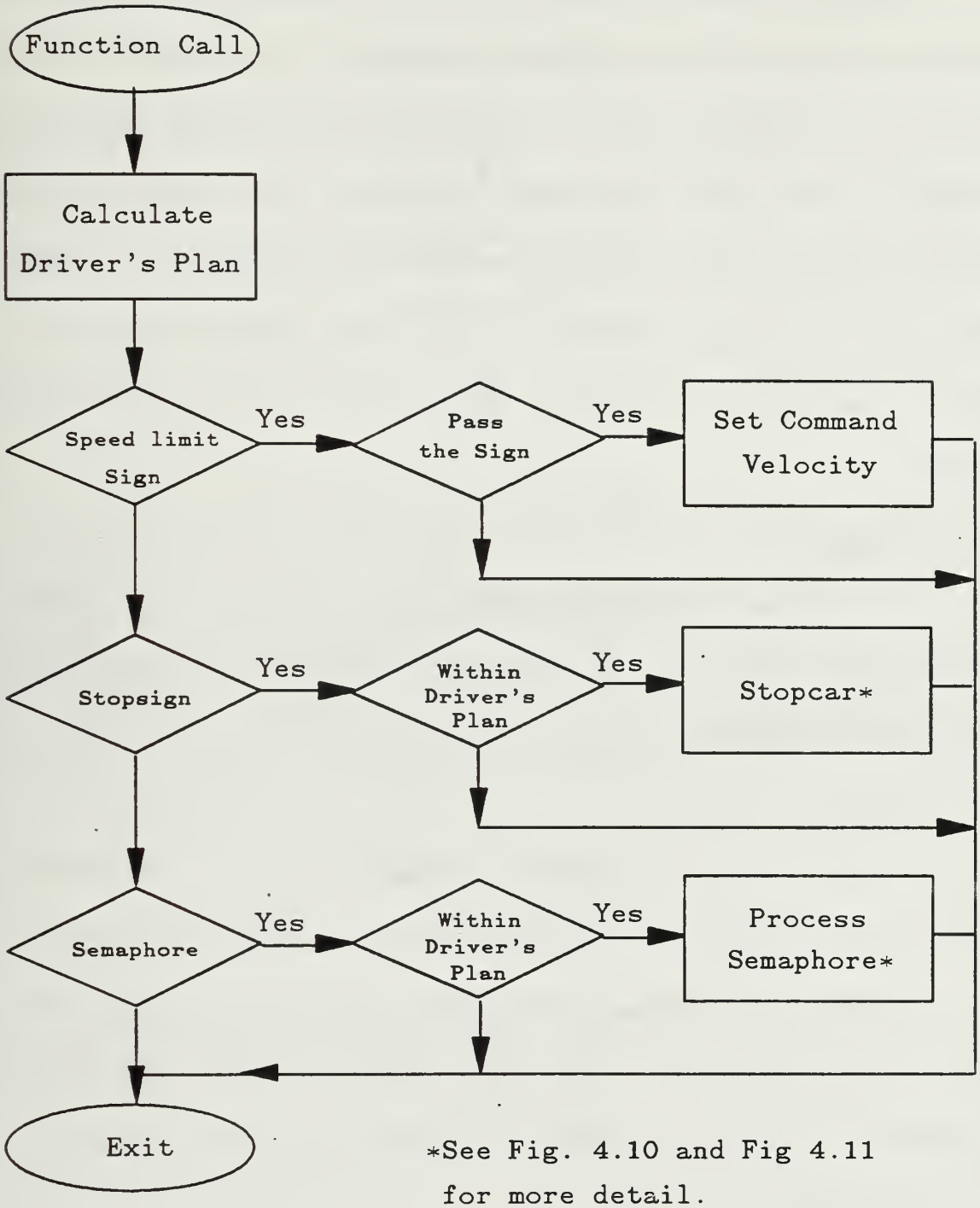


Figure 4.9 CRUISE.C flowchart

3. Brake.c

Brake.c has several functions, but is also instantiated only during the cruise control modes. First, this module calculates d_p, the driver's perceived distance to the stop sign using Eq. (3.3) and the vision theory developed in Chapter III of this work. Also, brake.c computes v_p, the driver's perceived velocity using Eq. (3.6). Finally, the module provides a_{brake}, the braking deceleration for the vehicle utilizing Eq. (3.28). Engine braking, as defined in Eq. (3.18), is computed in the vehicle simulation. Fig. 4.10 is a flowchart of this module.

4. Signal.c

This module is also instantiated in the cruise control modes. Signal.c provides vehicle control input for the different light phases of the semaphore in the highway environment as depicted in Fig 4.11.

5. Clear.c

Clear.c has only a single function. It is called in the cruise control modes after a complete stop at a stop sign. The module simulates a driver observing that it is safe to proceed through an intersection. Clear.c uses the system clock to delay an arbitrary length of time, then removes the brake and sets the commanded velocity to the last assigned speed limit on that stretch of highway.

6. Mapview.c

This module has two main functions. First, mapview.c builds an overhead view of the highway environment with its crossroads, stop signs and semaphores

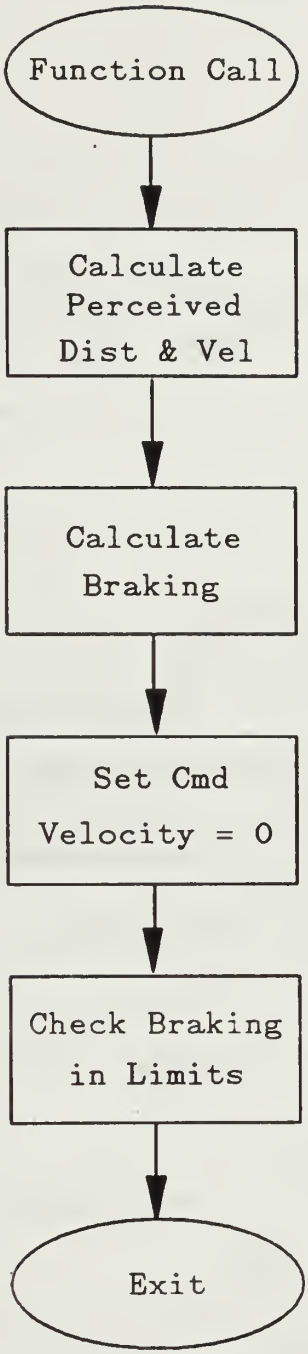


Figure 4.10 BRAKE.C Flowchart

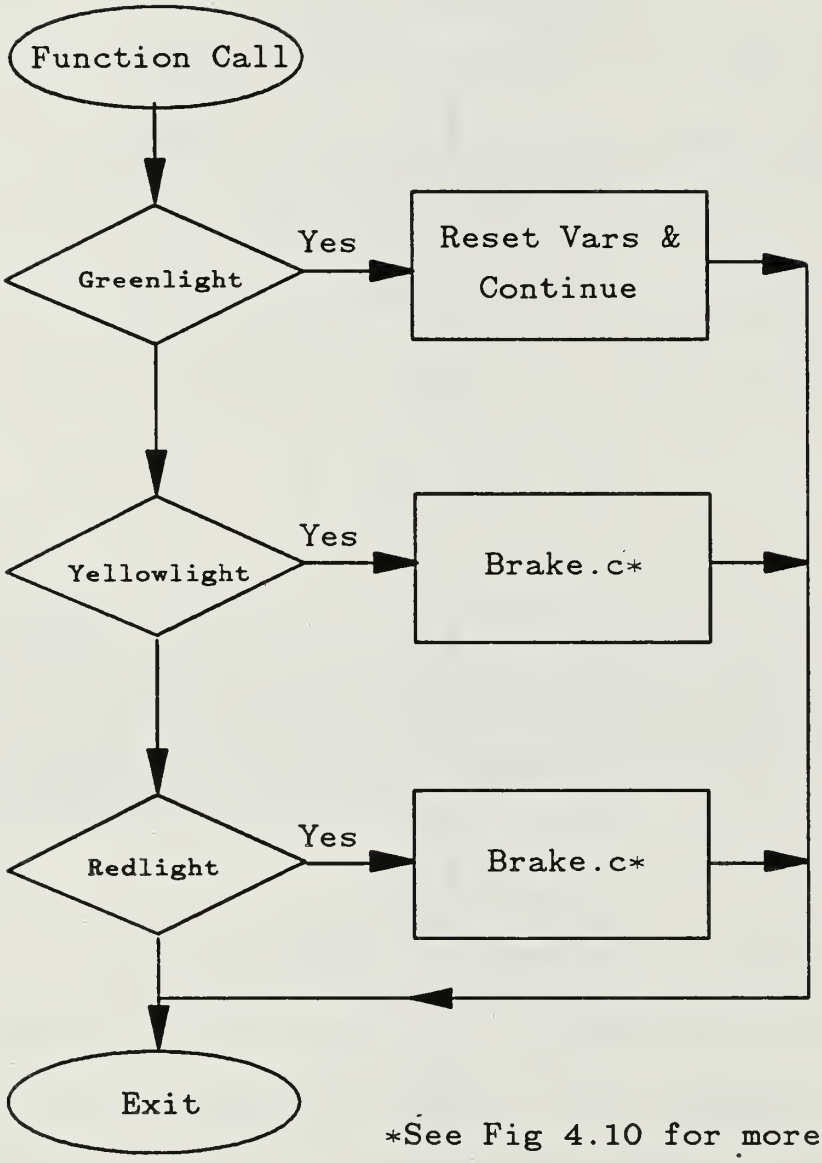


Figure 4.11 SIGNAL.C Flowchart

as shown in Fig. 4.2. Lastly, this module uses information received by navigate.c from the driver's display to plot the vehicle location by moving *crosshairs* around the road circuit.

7. Gauges.c

The function of gauges.c is to build the instrumentation and control placard shown on the right of Fig. 4.2. Data for the instruments is provided by navigate.c.

8. Mouse.c

The function of this module is to allow the user to manually govern the vehicle's speed and braking.

9. NetV.c

NetV.c was designed and implemented by Manley [Ref. 34]. Its purpose is to open an Ethernet connection to the driver's display on the other IRIS workstation.

10. Checkkey.c

This function monitors the keyboard, allowing the user to change driving modes with a single keystroke.

11. Savedata.c

Savedata.c stores test data which has been saved during program execution in temporary arrays. At program completion, savedata.c is invoked and writes the test data to a file named *test*.

12. Generate.c

This module uses the system random number generator and provides output in the form of Gaussian random numbers. The IRIS's random number generator is seeded from the system clock to ensure unique output of uniformly distributed random numbers during each execution of the simulation. Then, a Gaussian distribution can be closely approximated by summing twelve random numbers uniformly distributed between -1 and $+1$ and dividing the total by two.

13. Loadintarray.c

The function of loadintarray.c is to read vision information from a file named *roadmap* to an array during program initialization. The vision information is later used by cruise.c during analysis in the cruise control modes.

14. Welcome.c

The purpose of this module is to display a welcome banner on the IRIS monitor while the graphics system is initialized and files are loaded.

15. Const.h

User defined constants for the navigator's display have been organized in this file to make future software modification easier. Const.h must be an include file in each program module which contains program constants.

16. Vars.h

All global variables for navigator's display are in this file. Vars.h must be listed as an include file only in the navigator.c module.

17. Vars.ext.h

External variables which are required when working in the C programming language are in this file. Vars.ext.h must be listed as an include file in all modules other than navigator.c

18. Vision.h

This file contains simulated vision input required for program operation. The locations of stop signs, speed limit signs, and the semaphore are provided. Additionally, for each speed limit sign, there is an associated speed in the file.

19. Makefile

This is a utility for program organization and management provided with the UNIX operating system. It assists the user by keeping track of which files need to be recompiled following modification. [Ref. 36:p. 105]

G. MODULE DESCRIPTION FOR THE DRIVER'S DISPLAY

Most of the modules on the driver's display were designed and implemented by McGhee, Zyda, and Tan [Ref. 30:pp. 46-67]. However, many modifications were made to support this work. Therefore, all modules on the driver's display are discussed below in depth.

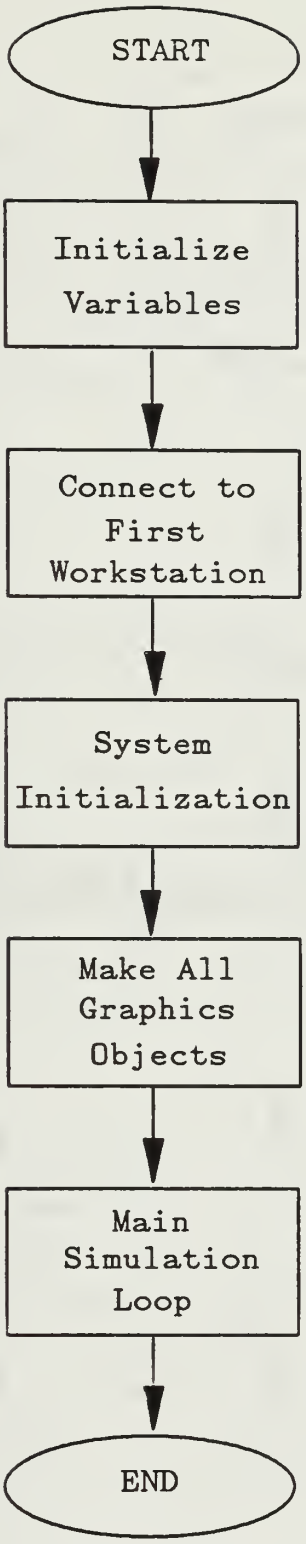
1. Carsimu.c

This is the control module for the driver's display graphics simulation. Additionally, carsimu.c has several important functions. First, it initializes the IRIS graphics system, sets all local and global variables, and completes the

connection to the navigator's display on the other workstation. Carsimu.c also transmits pertinent vehicle data to the navigator's display and receives command and control information in return. While in the autosteer modes, this module calculates the headings to the successive target points in the center of the vehicle's driving lane. Carsimu.c also processes all manual steering input from either IRIS workstation's mouse. Lastly, it processes manual longitudinal speed control input from the driver's display. Fig. 4.12 through Fig. 4.18 is a flowchart of this module.

Several changes have been made in this module to support this research. First, the target steering system implemented in earlier research is dependent on high, constant velocities and becomes unstable at low speeds or during complete stops. Therefore, the *AutoSteer* modes cannot be used at velocities near zero. Thus, this module has been changed to default to a *Manual* mode whenever the velocity decreases to approximately 3 kph. This logic is displayed in the flowchart of Fig. 4.14. Additionally, checkkey.c, a routine which scans for keyboard input, is designed to accept *AutoSteer* modes only when the vehicle velocity exceeds 11 kph.

Lastly, for this work, the authors desired the vehicle be capable of multiple laps on the closed-course track. This requires an improvement to overcome the problem of discontinuity in the arctangent function used for the *AutoSteer* modes [Ref 30:p. 66]. An implementation of the algorithm is shown in Fig. 4.19. In this solution, σ_i is saved and used as a reference in the calculation



(See Figure 4.13 for more detail)

Figure 4.12 CARSIMU.C Flowchart

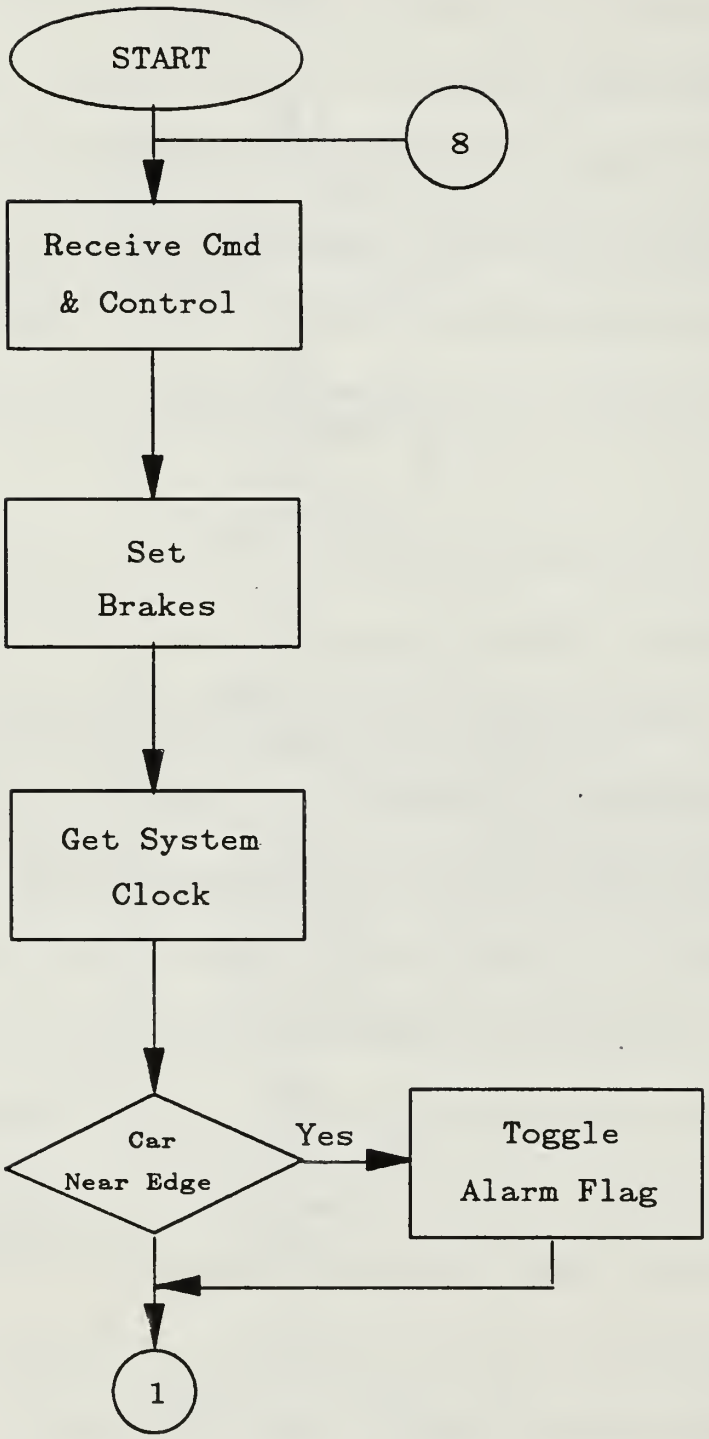


Figure 4.13 Main Loop of Carsimu.c (Part 1)

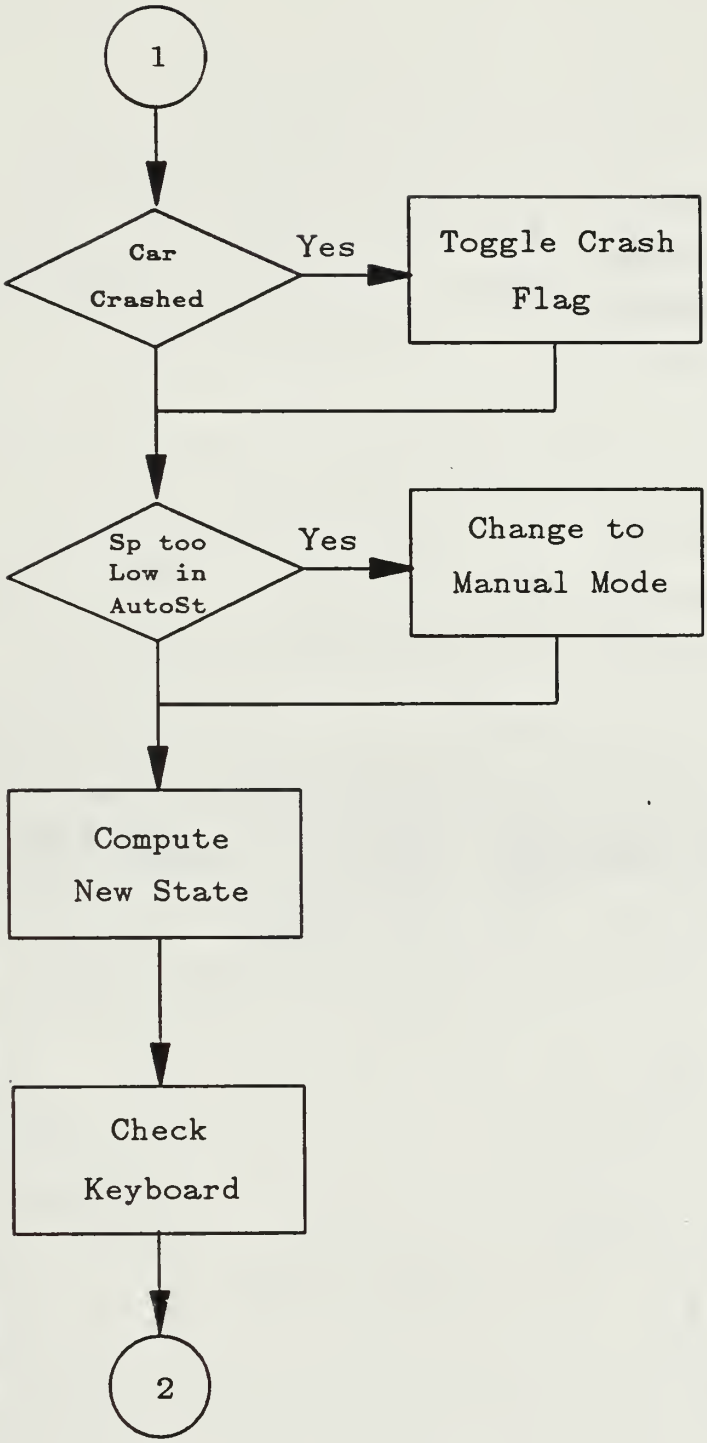


Figure 4.14 Main Loop of Carsimu.c (Part 2)

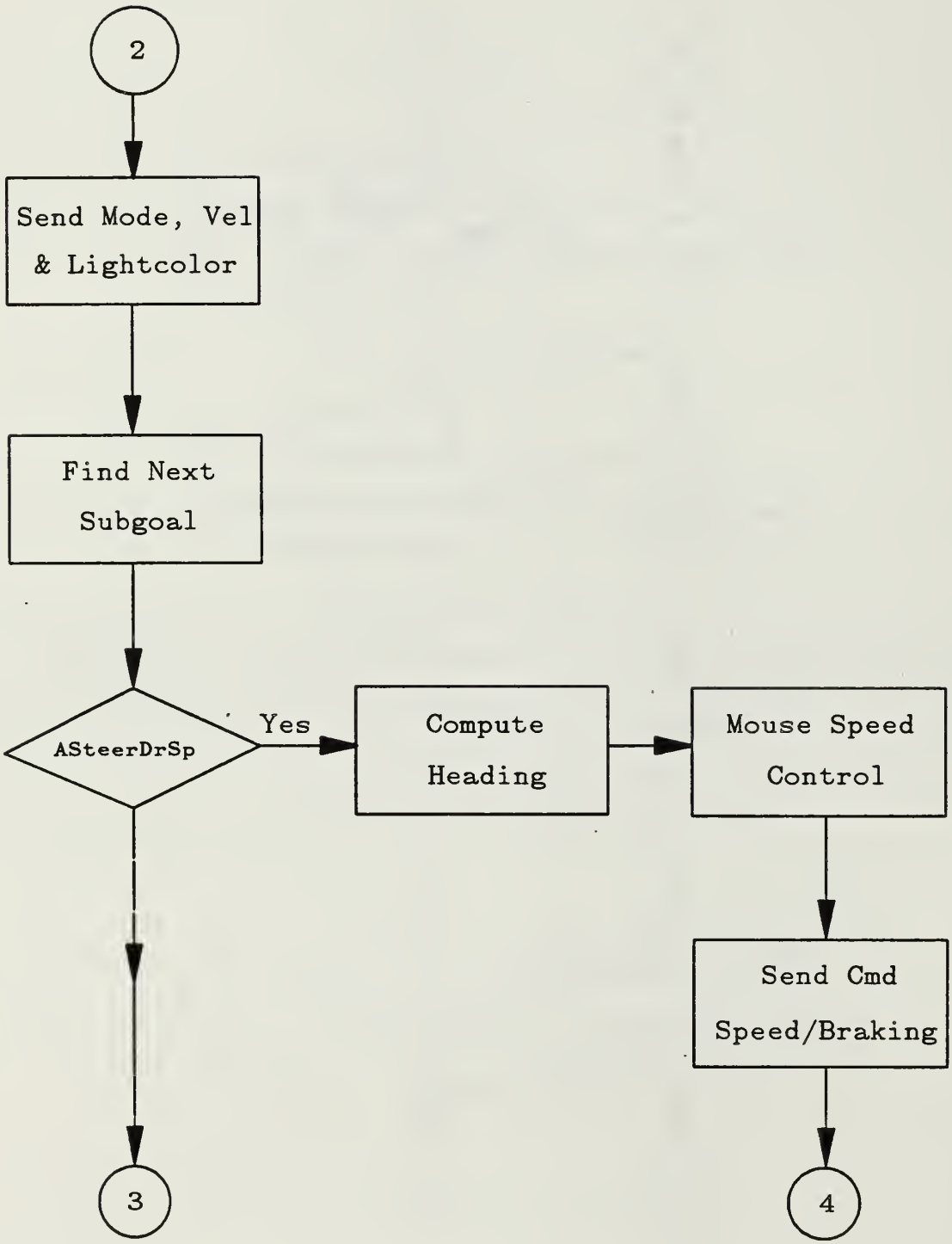


Figure 4.15 Main Loop of Carsimu.c (Part 3)

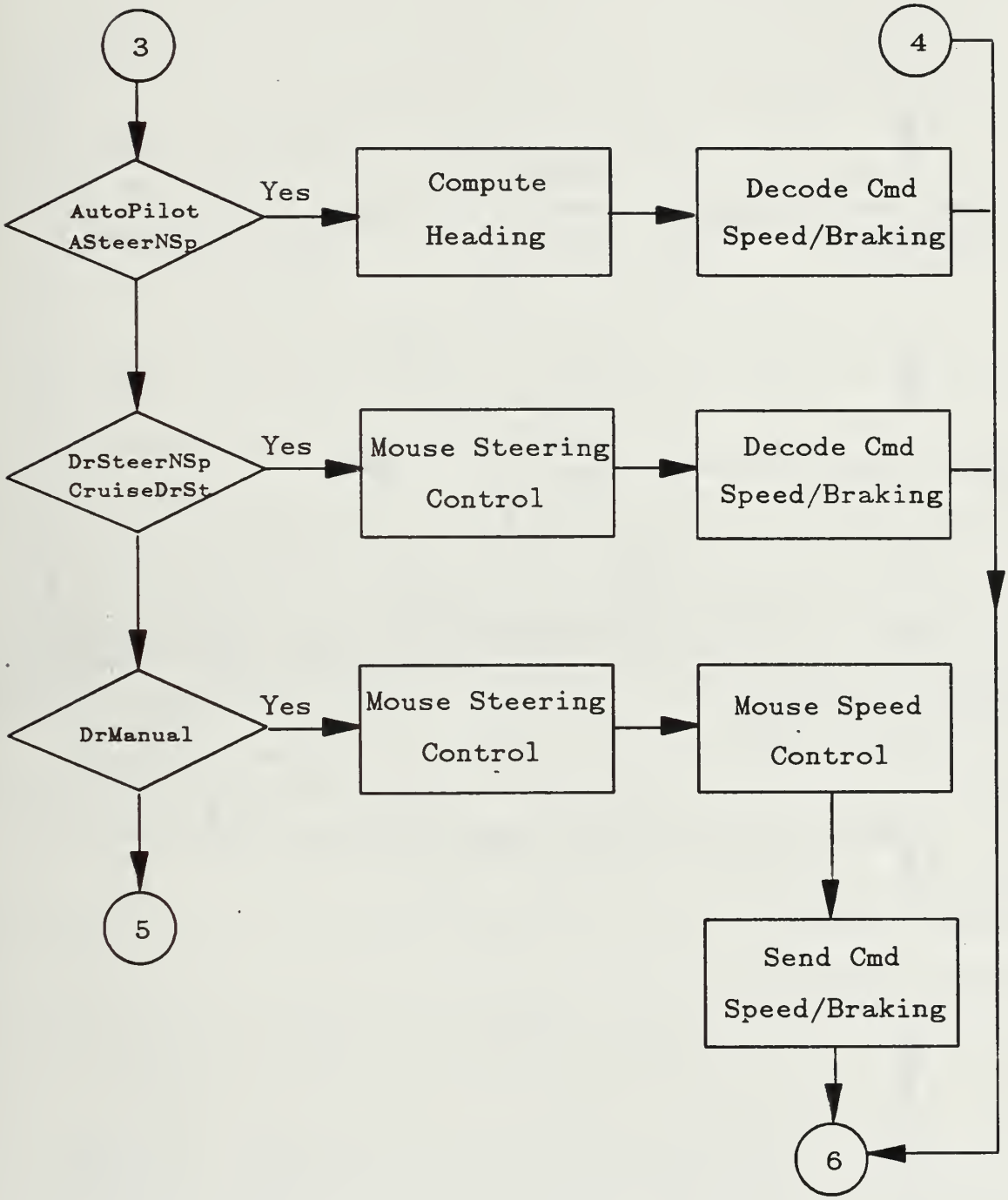


Figure 4.16 Main loop of Carsimu.c (Part 4)

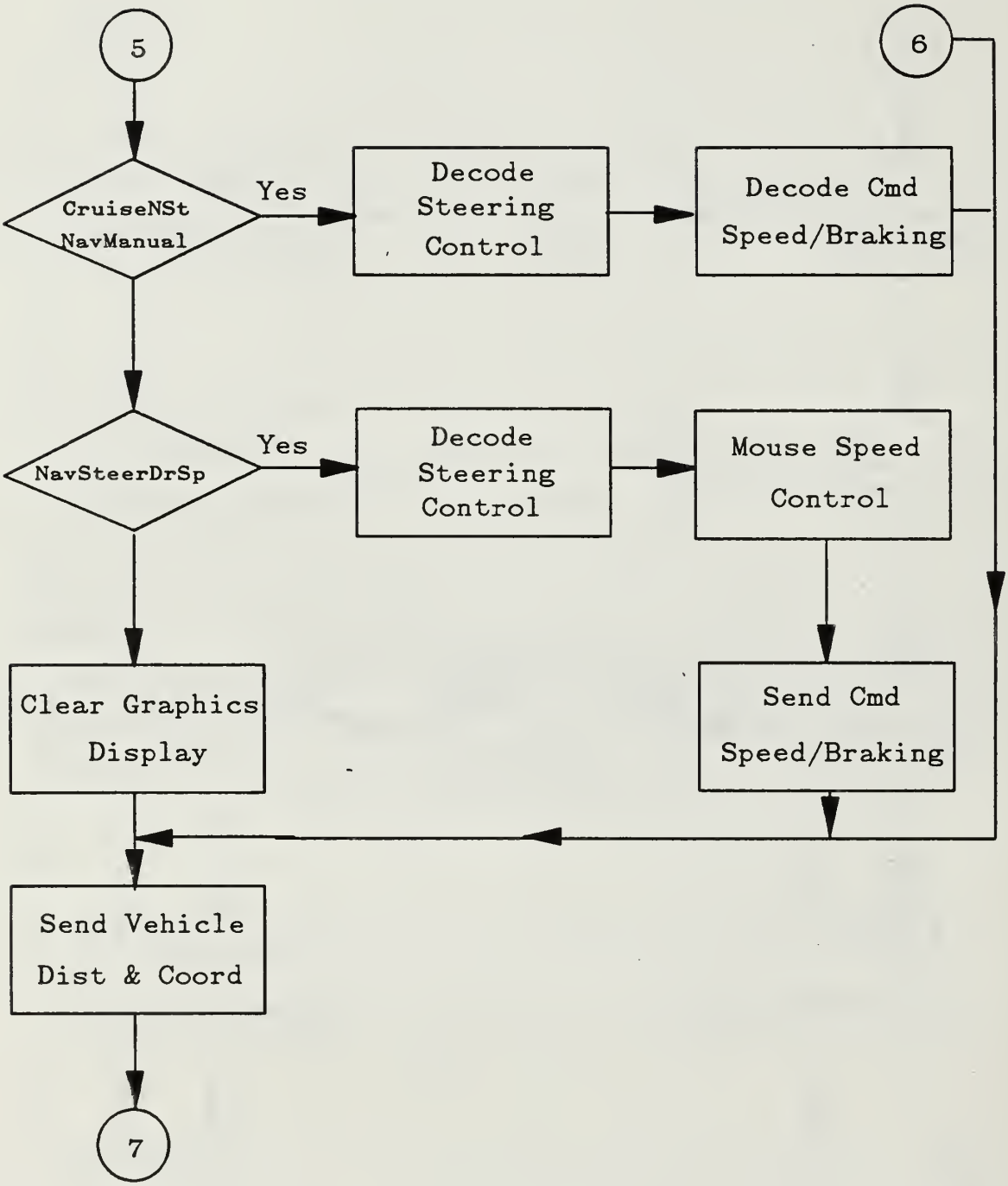


Figure 4.17 Main Loop of Carsimu.c (Part 5)

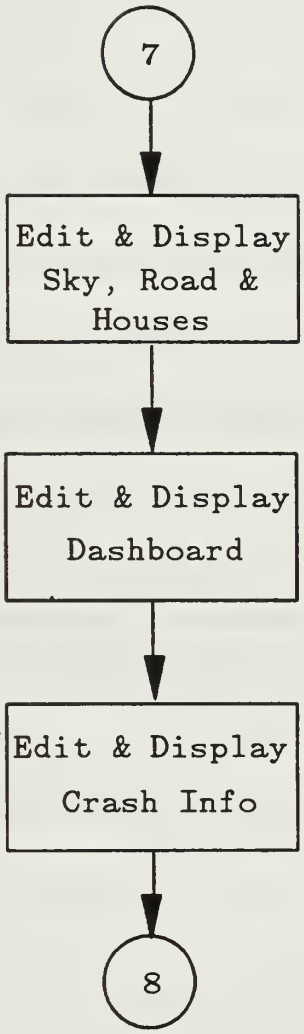


Figure 4.18 Main Loop of Carsimu.c (Part 6)

of σ_{i+1}. If σ_{i+1} is significantly different from σ_i, a discontinuity exists and σ_{i+1} is adjusted. In this work a variation of 3.5 radians is allowed for routine corrections and normal engagement of the *AutoSteer* modes. When a discontinuity is encountered, a variation of approximately 2π radians is observed.

2. Circuit.c

The primary function of circuit.c is to build the highway environment for the vehicle simulation. Several modifications to support this work were incorporated since its design. First, a fourth curve has been added to close the rectangular circuit as shown in Fig. 4.2. Additionally, two crossroads have been added along with three stop signs, several speed limit signs, and a semaphore.

```
sigma = atan2((gx - cx),(gy - temp));
if (sigma - last sigma < - 3.5)
        {
        lap = 1 + (int)((lastsigma - sigma - PI)/(2 * PI));
        sigma = sigma + 2 * PI * lap;
        lastsigma = sigma;
        }
        else lastsigma = sigma;
sigma_dot = (sigma - old_sigma)/deltat;
old_sigma = sigma;
```

Figure 4.19 Key C-code for the Arctangent Discontinuity.

3. Other.c

This module contains the support routines used to build the sky, mountains, semaphore, stop signs, and other terrain features in the driver's display.

4. Find subgoal.c

When in an autosteer mode, find_subgoal.c searches for the next target point to steer towards. The module is designed such that the target point is continually being computed. However, the information is only used when in an autosteer mode. This is done for two reasons; first, to ensure the target point is always in front of the vehicle and second, for efficiency when an autosteer mode is engaged.

5. Integrate.c

This module performs the Euler-Heun numerical integration required to provide vehicle dynamics [Ref. 30:pp. 64-65]. In the autosteer modes, the steering angle for the dashboard display is computed. In the manual modes, the module allows the driver to control the steering wheel angle.

6. Display.c

The dashboard display is built by this module as shown in Fig. 4.1.

7. Checkkey.c

This module responds to all input from the keyboard, through which the user can change driving modes.

8. Welcome.c

Welcome.c displays a welcome banner while the IRIS graphics system is initialized and files are loaded.

9. Letter.c

This utility was designed and implemented by J. Artero and R. Kirsch, and was later modified by L. Williamson. The module creates most of the upper case Roman alphabet for use in graphics displays. Several numbers have been added to create speed limit signs for our work.

10. NetV.c

The function of this module, designed by Manley [Ref. 34], is to establish an Ethernet connection to the navigator's display on the other workstation.

11. Loadarray.c

Loadarray.c reads a file named *roadmap* containing the target points which the vehicle steers toward while in the autosteer modes. For efficiency, the points are stored in an array during program execution.

12. Roadmap

This is the file containing the target points which the vehicle steers toward while in the autosteer modes. The points are one meter apart and match the center of the vehicle's driving lane.

13. Const.h

Programmer defined constants for the driver's display are organized in this file to make software modification easier. Const.h must be an include file in each program module containing programmer defined constants.

14. Vars.h

All global variables for carsimu.c are in this file. Vars.h is an include file only in carsimu.c.

15. Vars.ext.h

Vars.ext.h contains all the external variables required when programming in the C language. All modules, other than carsimu.c, must list vars.ext.h as an include file.

16. Map.c

This routine is independent of the other modules on the driver's display. Map.c is used to generate the *roadmap* file and the target points the *roadmap* file contains.

17. Makefile

This is a utility for program organization and management provided with the UNIX operating system used on the IRIS workstations. Its purpose is to assist the user by keeping track of which files need to be recompiled following modification. [Ref. 36:p. 105]

H. USER'S GUIDE

To execute the graphics simulation, enter the following commands. First, on the driver's display workstation enter the command:

carsimu

The display will respond with:

Server waiting to connect to npscs-iris1

This response indicates the driver's display has opened a socket to the second workstation. At this time, on the navigator's display enter the command:

nav

The navigator's display will complete the connection to the other workstation and graphics initialization will begin. It takes a short time for the workstations to read the roadmap into memory and make all the graphics objects.

The driver's display begins as shown in Fig. 4.1. The top portion of the IRIS monitor is the "out-of-the-windshield" view of the highway environment and the lower portion of the display is the vehicle instrumentation and a control placard.

The navigator's display, shown in Fig. 4.2, has an overhead view of the highway environment on the left portion of the IRIS monitor. On the right side of the monitor pertinent vehicle instruments and a control placard are displayed.

A user can change operating modes by referring to the control placard and providing the appropriate input on the keyboard of either workstation. Fig. 4.3 provides more detailed information on the operating modes. To exit, enter an e on either keyboard.

To control the speed of the vehicle manually, select the proper mode (see Fig 4.3). Then, clicking the rightmost mouse button corresponds to incrementally depressing the accelerator in a vehicle while holding the right mouse button corresponds to steadily increasing the accelerator depression in a vehicle. Clicking the center mouse button corresponds to incrementally releasing the accelerator and holding the center mouse button corresponds to steadily decreasing the accelerator. A user can release the accelerator immediately by clicking or holding the left mouse button. The brakes can be applied by displacing the mouse vertically (away from the user). To remove or reduce the braking, move the mouse toward its original position (toward the user). Instrumentation provided on each display and the "out-of-the-windshield" view assist the user.

To steer the vehicle while in one of the manual steering modes, move the mouse to the left or the right, as necessary. The steering wheel turning rate and displacement on the driver's display is proportional to the speed and displacement of the mouse. Again, the instrumentation on each display and the "out-of-the-windshield" view assist the user. Additionally, if the vehicle is too close to the

edge of the highway, the **Danger** warning indicator on the driver's display will begin to flash. If the vehicle should leave the highway and crash, the user must exit the simulation by entering **e** on the keyboard.

Test data can be recorded by entering a **t** on the navigator's display. Because of the volume of information stored, it is recommended the this feature be activated only when conducting a trial simulation. The recorder can be toggled off by entering **t** a second time. At program termination, the data is permanently saved in a file named *test*.

I. SUMMARY

The hardware and software for the three-dimensional graphics simulation model are discussed in this chapter. A complete system has been developed through which experiments can be conducted involving both manual and computer controlled longitudinal speed control. In the next chapter, experiments using this system are described and analyzed.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

Numerous simulation trials were conducted with different program variables to check the validity of the stated hypotheses and the correctness of the mathematical model derived in Chapter III. The results of the simulation trials were recorded and plotted for analysis and documentation.

The closed-course track as shown in Fig. 4.2 has three stop signs and a semaphore. However, for these studies, only the semaphore and stop sign #2 were used. This allowed the driver or the cruise control system adequate time to stabilize at the maximum safe speed for that stretch of road prior to encountering a stop sign or a semaphore with a red light. Additionally, this work studies only speed control and not the more complex analysis required when approaching an intersection as the semaphore turns yellow. Thus, for these trials, the simulation was set such that the semaphore is always red as the vehicle approaches.

In all experiments, the trial begins by setting and stabilizing the velocity at the desired value. As the vehicle approaches the red semaphore or stop sign #2, the driver or cruise control adjusts the vehicle speed and uses braking as necessary so as to stop at the appropriate location. The trial is complete when the vehicle comes to a stop.

Several trials have been made to record the characteristics and results of manual and cruise control braking. In each case, regression analysis employing a least-squares fit has been applied to relate the deviations resulting from several stops [Ref. 37]. Using the resultant curve, $\ddot{x}$, the total acceleration due to braking can be determined. A value for a_{max}, the maximum acceleration due to braking can be determined by using the results obtained from a trial with 100% braking. This is shown in Fig. 5.1. Now, the driver's acceleration and deceleration plan can be calculated with

$$\alpha = \frac{\ddot{x}}{a_{max}} \tag{5.1}$$

The remainder of this chapter documents the results of several experiments. First, the trials conducted with manual longitudinal speed control are reviewed and the results obtained from these experiments are discussed. Then, the experiments with the cruise control system, which incorporates the mathematical model developed in Chapter III, are discussed and analyzed.

B. MANUAL LONGITUDINAL CONTROL

During the manual longitudinal control trials, it was found to be extremely difficult to stop at the required location. This was attributed to a lack of visual references which help humans to perceive motion and accurately judge distances. Additionally, a time period was required for the driver to become accustomed to the dynamic characteristics of the vehicle. To help overcome these handicaps,

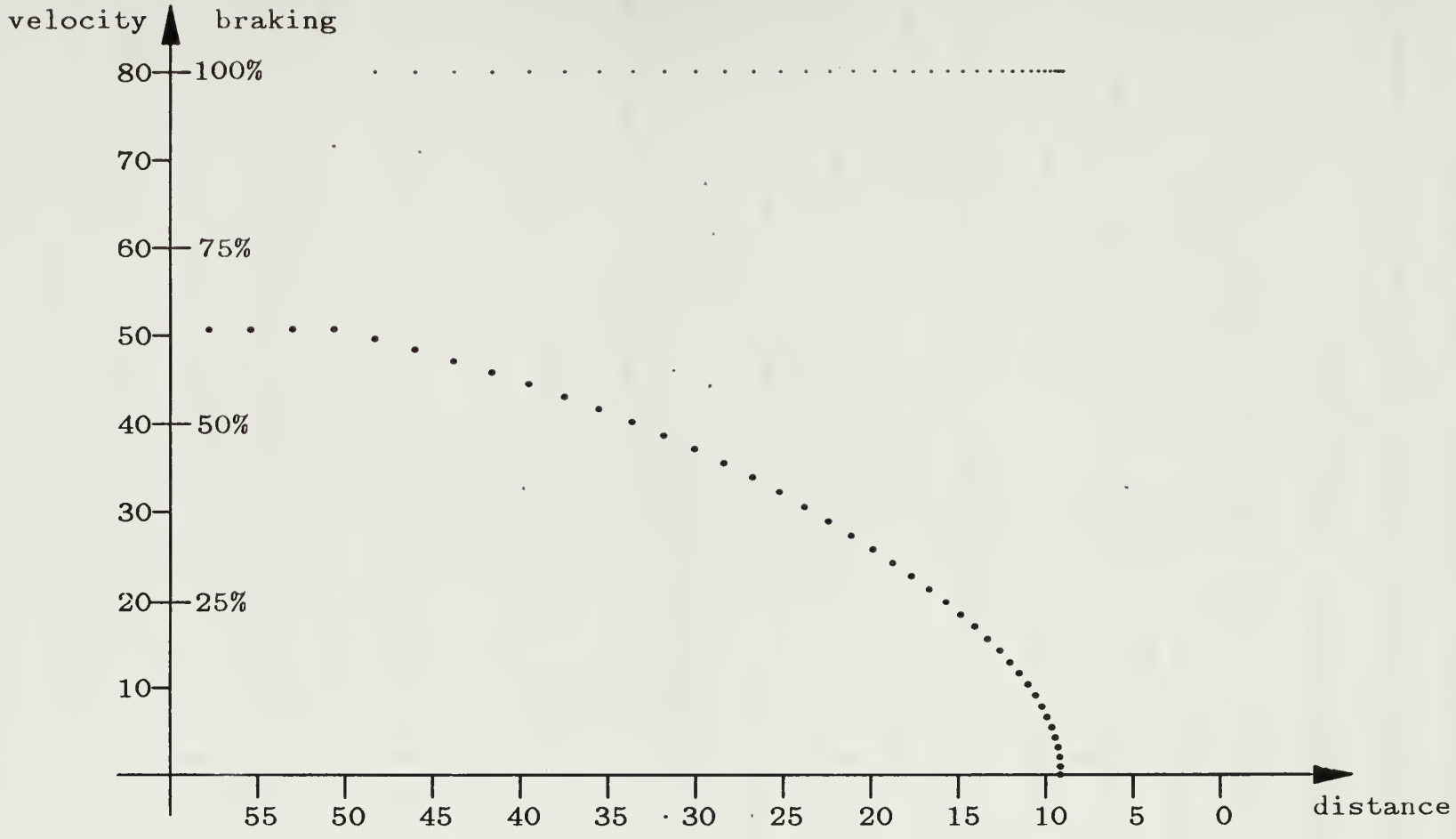


Figure 5.1 Maximum Braking provides acceleration of - 1.39 m/s**2.

distance remaining markers were added at ten meter increments for the last 60 meters approaching the semaphore. Also, the test driver practiced several stops in order to grow accustomed to the vehicle dynamics.

Typical results for manual longitudinal control are shown in Figs. 5.2 - 5.5. In the trials for Fig. 5.2 and Fig 5.3, Driver #1 and Driver #2 attempt to stop at the semaphore while the operating mode is *NavSteerDrSp*. This mode is chosen because the driver is responsible only for speed control and not steering control, resulting in a reduced workload. In both cases, the trials were initiated from 50 kph. Fig. 5.2 and Fig. 5.3 show the difficulty encountered by the drivers when attempting to stop at a designated location. On occasion, one driver *overshoots* the semaphore and enters the intersection. Analysis of Driver # 1's data shows his α is 0.59 and for Driver #2, α is 0.76.

The trials for Fig. 5.4 and Fig. 5.5 were initiated at 75 kph but the operating mode was *ASteerDrSp*. This mode also relieves the driver of steering control which is difficult at 75 kph [Ref. 30:p. 70]. Data for Fig 5.4 and Fig 5.5 were recorded while approaching stop sign #2. To stop in this situation, the driver must intercept his acceleration and deceleration plan while the vehicle is traveling around a curve. Also, this stretch of highway does not have distance remaining markers, making it more difficult for the driver to judge the distance to the stop sign. Fig. 5.4 and Fig 5.5 show that on occasion both drivers fail to stop until in the intersection. For Driver #1, α is 0.73. An analysis of the data in Fig 5.5 shows that Driver #2 has an α of 0.82.

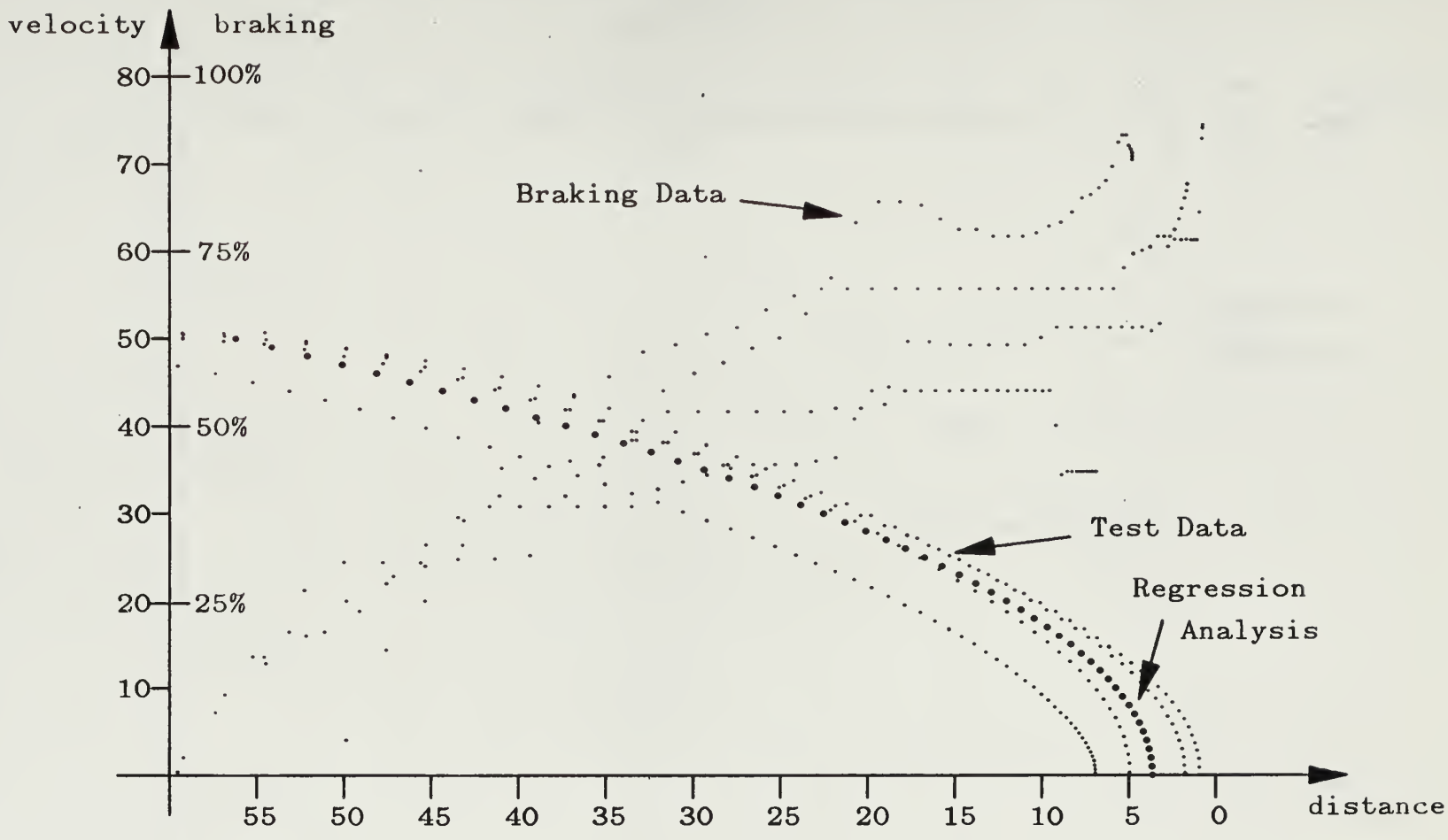


Figure 5.2 Four Trials by Driver #1, Alpha is 0.59.

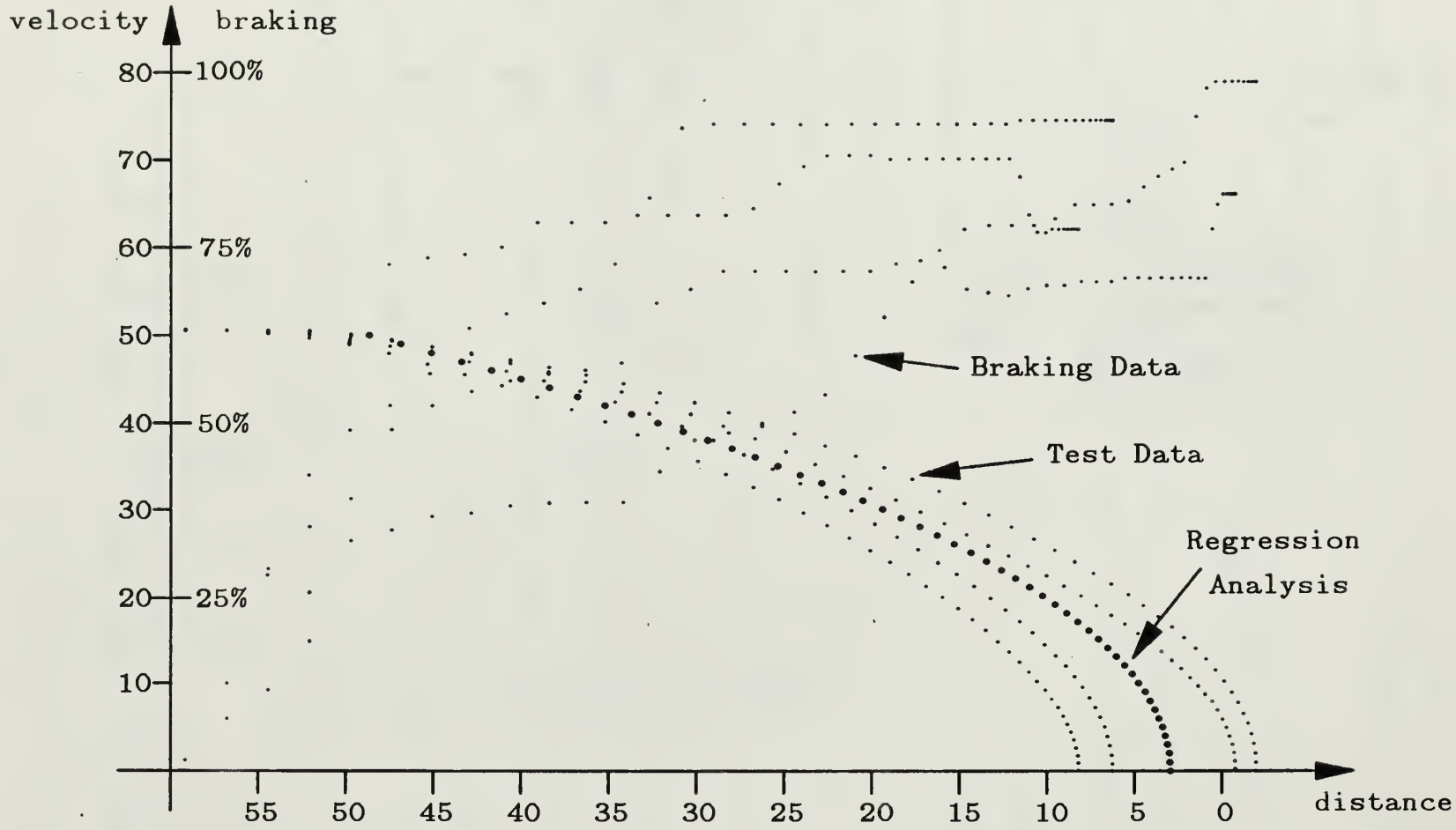


Figure 5.3 Four Trials by Driver #2, Alpha is 0.76.

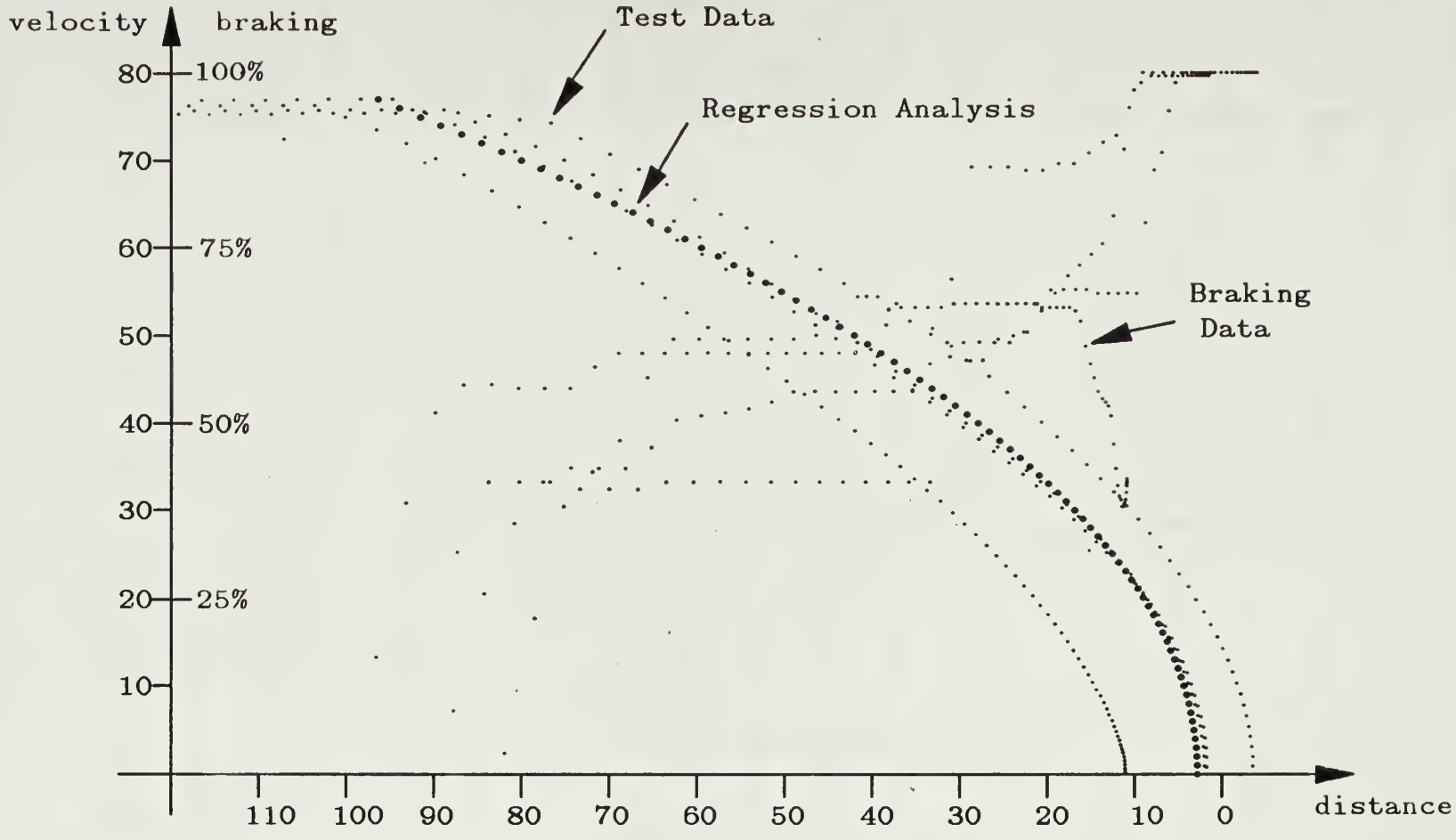


Figure 5.4 Four Trials by Driver #1, Alpha is 0.73.

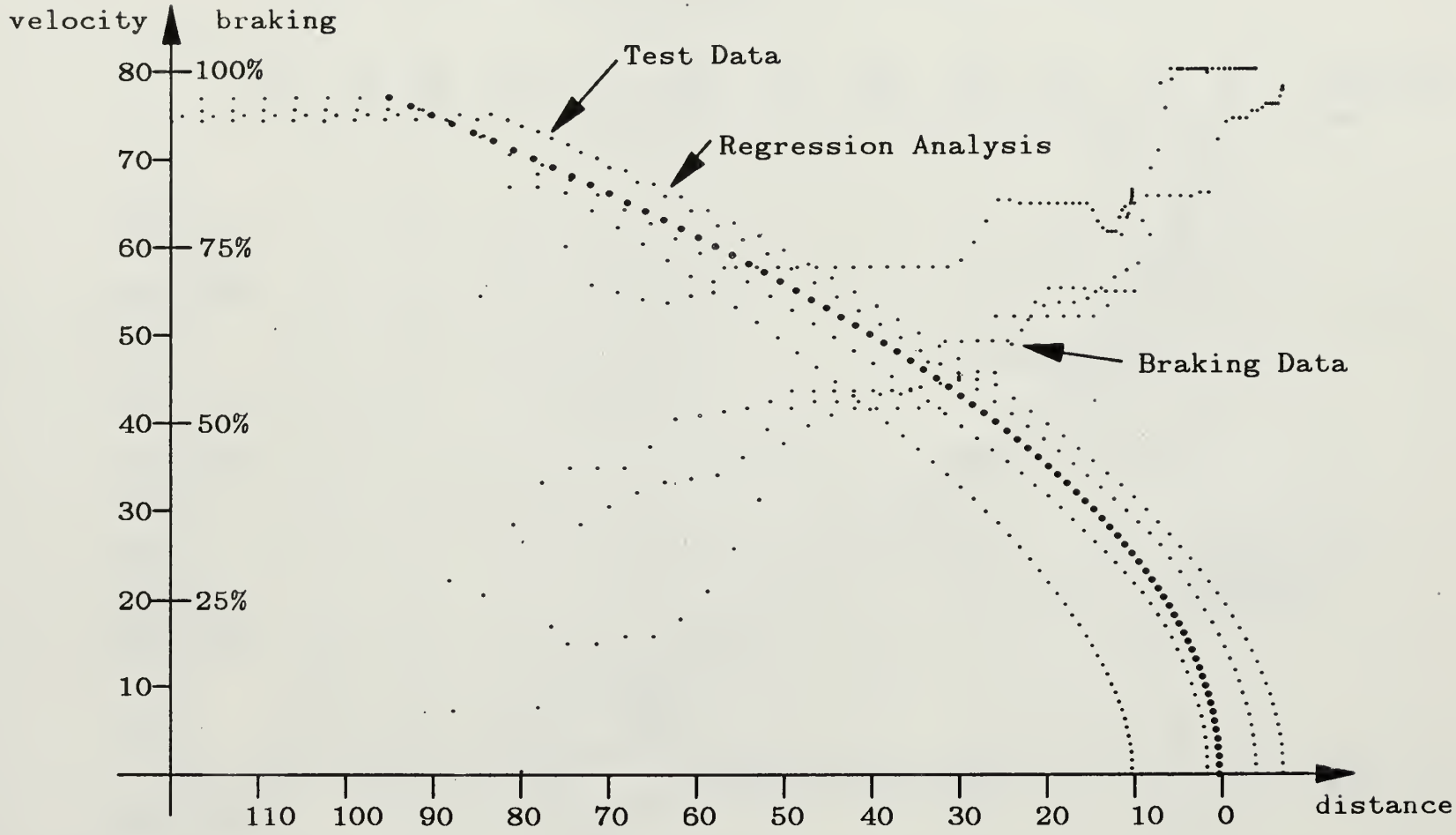


Figure 5.5 Four Trials by Driver #2, Alpha is 0.82.

C. AUTOMATIC DRIVING

The most difficult portion of the cruise control trials was finding a position gain factor, k_p, and a velocity gain factor, k_v. Note that for this work, two equations, Eq. (3.33) and Eq. (3.34), have three unknown variables. From the closely related work of McGhee, Zyda, and Tan [Ref 29:p. 39], λ is expected to be small, and most probably in the interval $-2.0 \leqslant \lambda \leqslant -0.5$. Therefore, values for k_p and k_v were calculated for $-3.0 \leqslant \lambda \leqslant 0.0$ in increments of 0.1. The acceleration and deceleration plan gain factor, α, was set to 0.5. To calibrate the system without the effects of noise resulting from a human driver's judgement, k_d, the distance multiplier, and k_s, the speed multiplier were both set to 1.0. Trials were then conducted for various λ within the prescribed domain. Results of trials at high velocities show for $\lambda > -2.0$, the vehicle stops short of the desired location. Subsequent trials with $\alpha = 0.6$ and $\lambda = -2.2$ showed the vehicle was capable of stopping properly from various speeds. Fig. 5.6 depicts the vehicle stopping from 50 kph and Fig. 5.7 displays the vehicle stopping from 75 kph.

With the system calibrated, the human driver is modeled by constructing k_d and k_s using input from the Gaussian random number generator. Fig. 5.8 is the result of four trials starting at 50 kph. It is of note that in all cases the cruise control *super driver* stopped the vehicle *prior* to entering the intersection. For the *super driver* in Fig. 5.8, α is 0.55. Similarly, Fig. 5.9 depicts four trials from 75 kph. Again, the *super driver* was successful in stopping the vehicle *prior* to

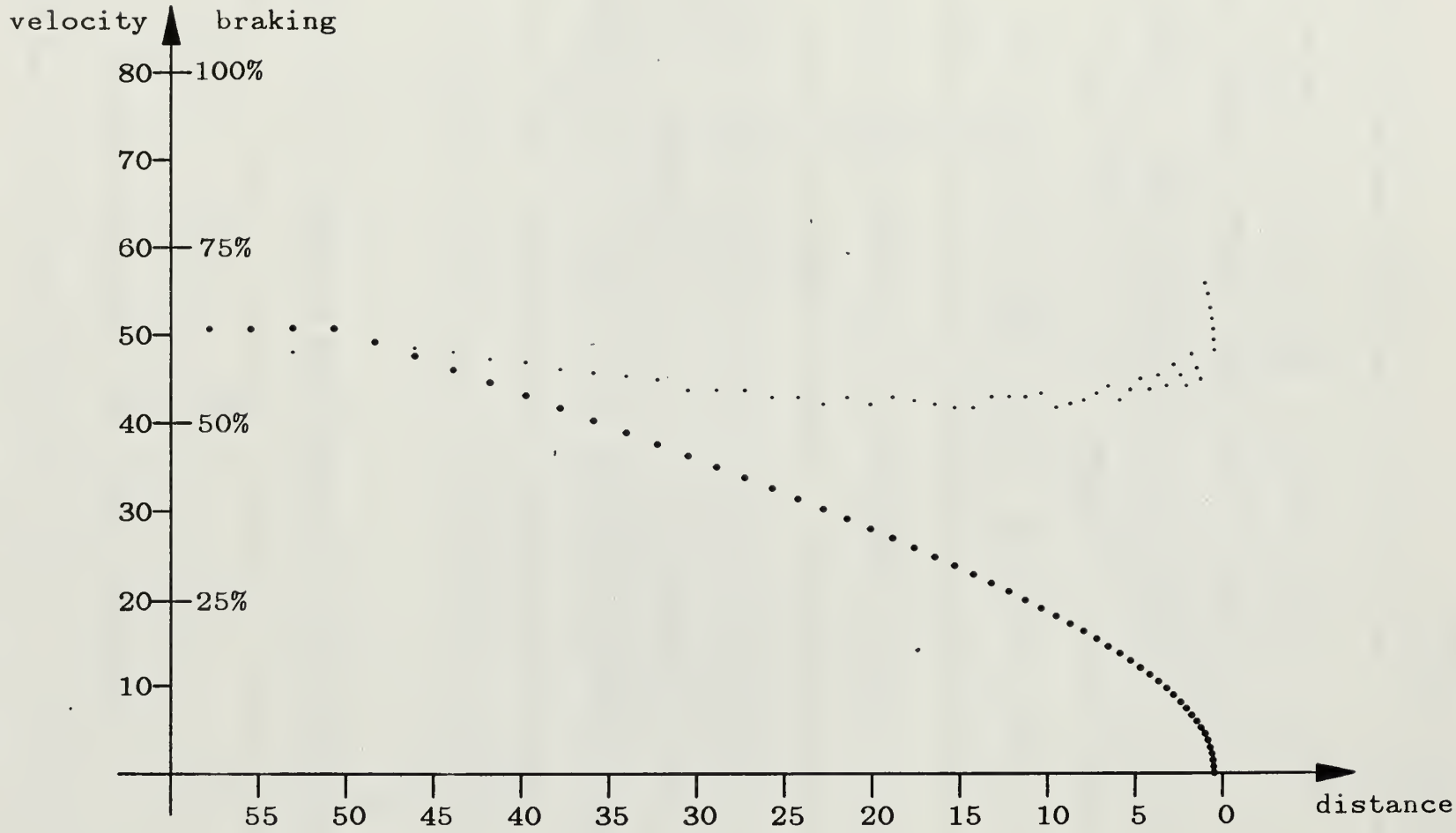


Figure 5.6 Calibration of the Vehicle at 50 kmph.

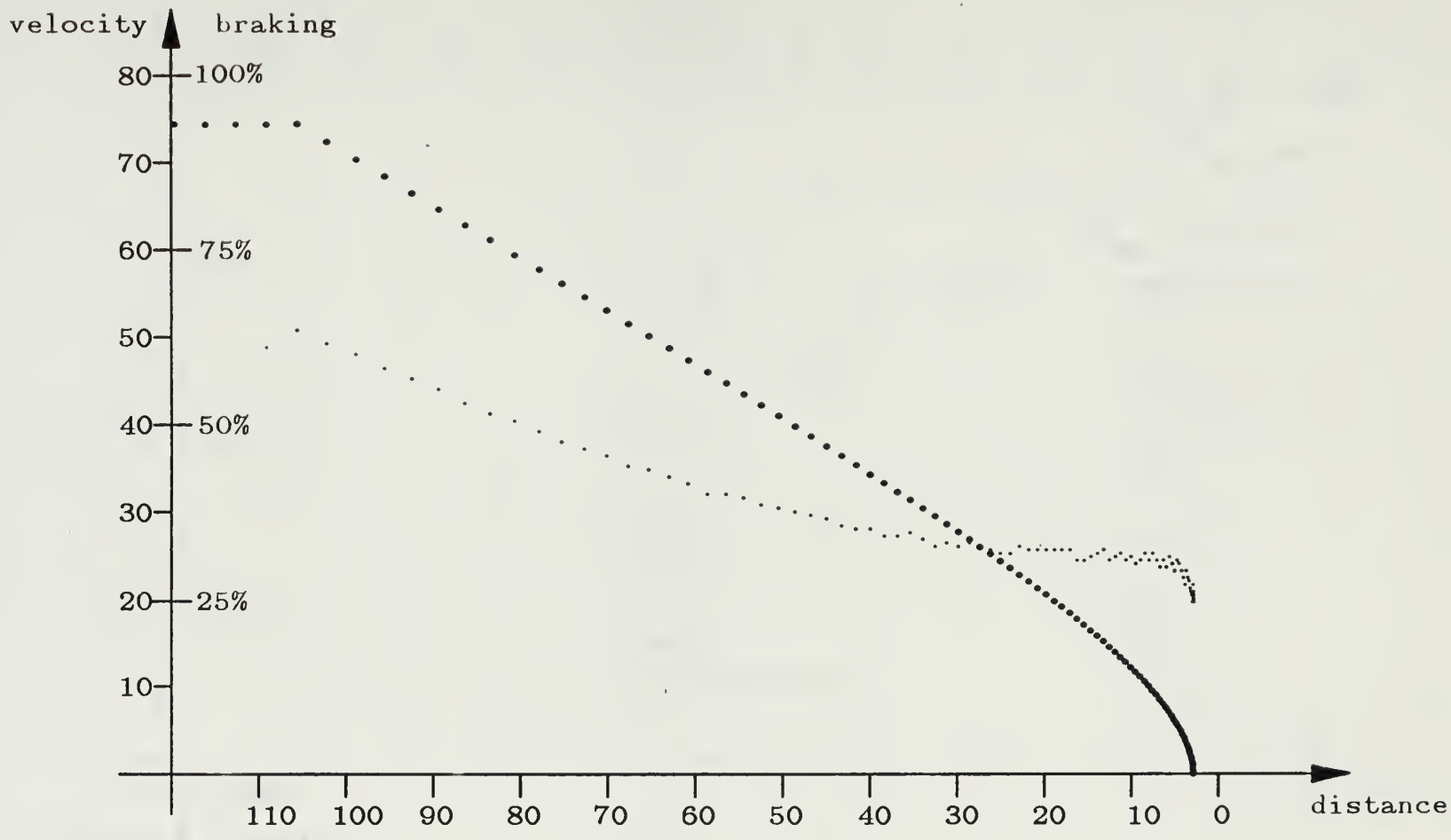


Figure 5.7 Calibration of the Vehicle at 75 kmph.

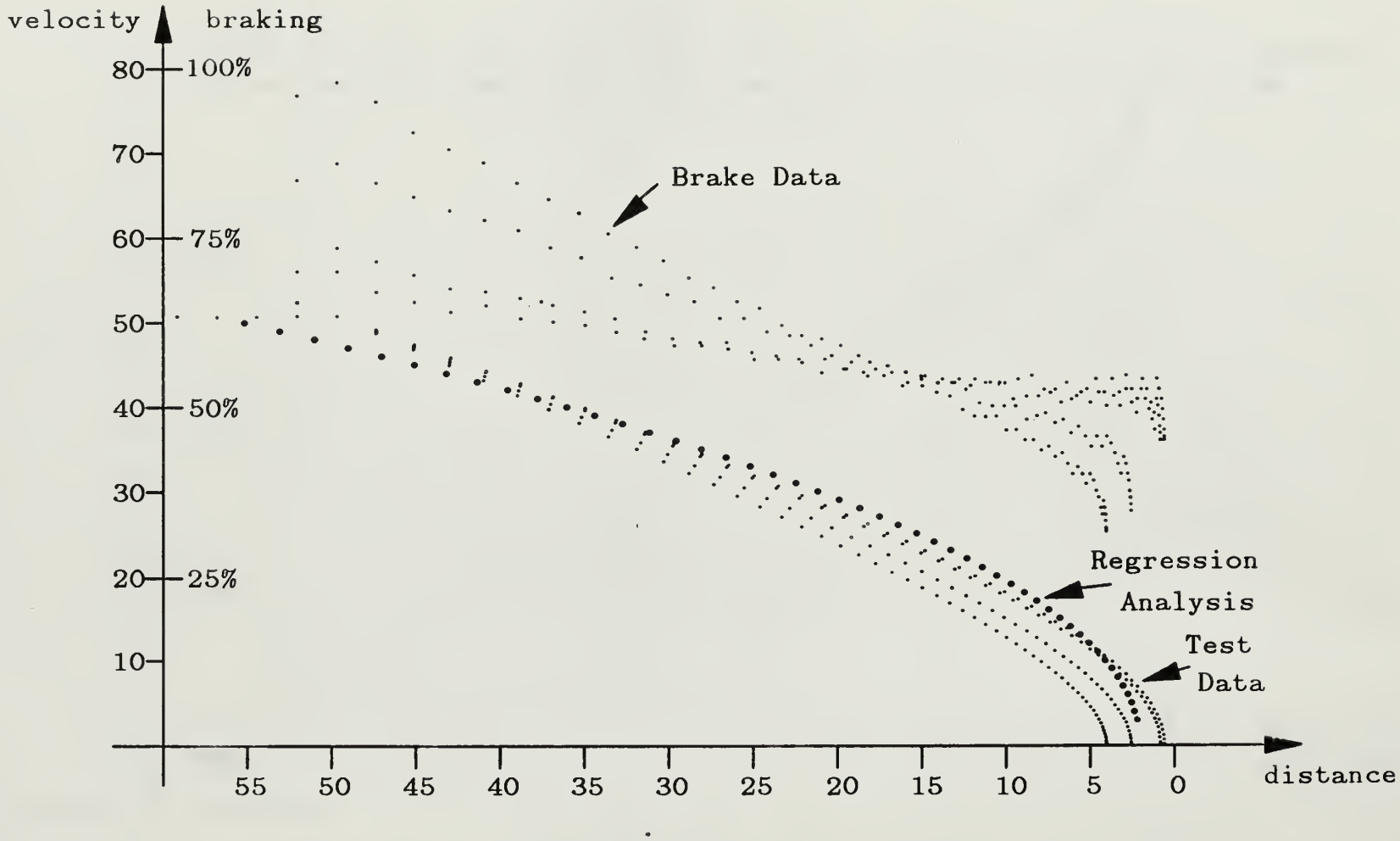


Figure 5.8 The "Super Driver" at 50 kmph.

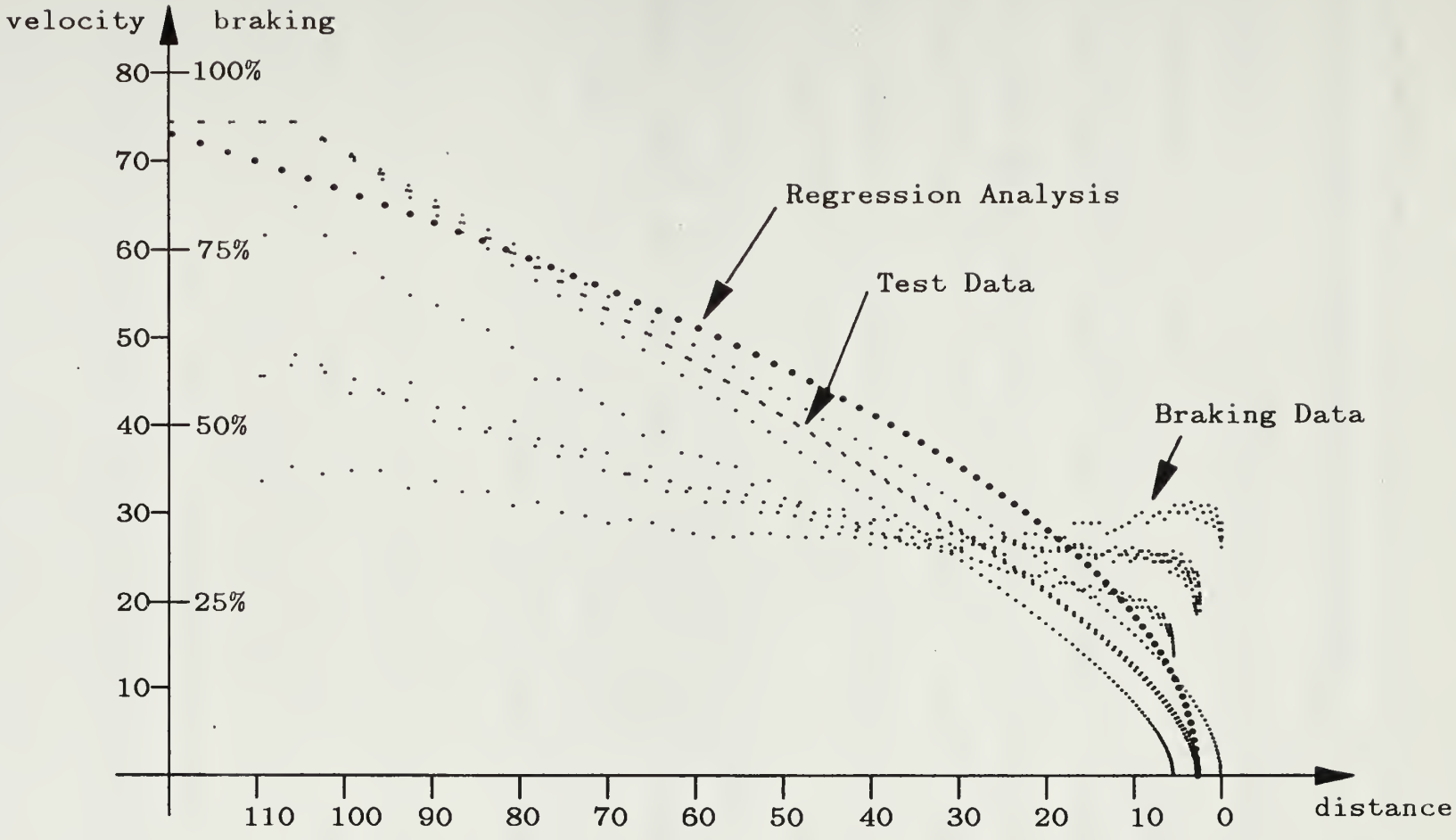


Figure 5.9 The "Super Driver" at 75 kmph.

entering the intersection. In this case, α is 0.53. Note that the autopilot applies hard braking early in the simulation trial, while the human driver uses hard braking late in the simulation trial. This may be due to the difficulty the human driver has judging the distance to the stop sign.

D. SUMMARY

The results of this chapter show that the behavior of a human driver as he estimates the distance to a stop sign and the velocity of his vehicle can be mathematically modeled. Additionally, the mathematical model developed in this work mimics this conscious or unconscious behavior to a significant extent. However, before any degree of confidence can be attached to the hypotheses of this work, additional research is needed to gather statistical data about how human acceleration and deceleration plans vary with driving conditions. Therefore, currently, the hypotheses of this work can provide at most a viable basis for longitudinal speed control in autonomous vehicles.

VI. SUMMARY AND CONCLUSIONS

A. SUMMARY

This work differs from previous research in the area of longitudinal speed control for autonomous vehicles in that the previous models did not mimic human control of the vehicle. Rather, much current research focuses on vision, sensors, planning, navigation, and avoidance. Few, if any, previous works explore the behavioral aspects of human driving which could provide some different insights into possible approaches to autonomous vehicle control.

At the start of this work, it was observed that human driving can be divided into two distinct levels, that of conscious and unconscious behavior. This work is concerned entirely with studying and modeling a simple conscious or unconscious aspect of human driving, that of controlling the accelerator and the brake position during a stop.

An important product of this work is the development of a three-dimensional color graphics simulation model utilizing two graphics workstations communicating over an Ethernet network. This model provides a realistic environment in which real-time experiments can be conducted to support ongoing research. The model can be modified, enlarged, or enhanced to facilitate related work in the future.

B. CONCLUSIONS AND POSSIBLE EXTENSIONS

In this simulation, the navigator's display is modularized for ease of management, modification, and expansion. Thus, several possible extensions to this research exist. First, a possible enhancement to the present system is a sophisticated vision model requiring complex vision analysis. The present model is simple and sees only key signs and semaphores. An improved model could possibly detect other traffic, obstacles, and even pedestrians. Such a model would require elaborate techniques and algorithms to detect and analyze each situation.

Additionally, the *Autosteer* mode may be improved to include automatic steering at low speeds and possibly at a complete stop. This extension would allow the user to conduct simulation trials without being distracted by mode changes.

Another possible extension to this work focuses on path planning and navigation. Such research would entail a complex highway environment or a cross-country environment with numerous routes. In this model, the navigator's display could select a route based on path planning and obstacle avoidance algorithms.

The extensions described above require a programming language and special hardware suited for advanced artificial intelligence applications such as vision, path planning, and obstacle avoidance. In consideration of these requirements, a final extension to this work involves replacement of the navigator's display and its host workstation by another system on a LISP machine. The use of the Ethernet communications in the implementation of this work allows an entire display and

its host computer to be readily replaced by a system such as a LISP machine. An implementation of the vehicle simulation under the control of a navigator's display on a LISP machine would provide an improved platform for advanced work in vision, path planning, and obstacle avoidance.

In conclusion, it is hoped that this research will serve as a basis and motivation for advanced work in the behavioral aspects of human driving. Research of this nature can have a significant impact on the development of autonomous vehicles of the future.

LIST OF REFERENCES

1. Todd, D. J., *Walking Machines, An Introduction to Legged Robots,* Kogan Page Ltd., London, England, 1985.

2. Osborne, D. M., *Robots, An Introduction to Basic Concepts and Applications,* Midwest Sci-Tech Publishers, Inc., Detroit, Michigan, 1983.

3. Thring, M. W., *Robots and Telechirs,* John Wiley & Sons, New York, New York, 1983.

4. Brady, M., Gerhardt, L. A., and Davidson, H. F., *Robotics and Artificial Intelligence,* Springer-Verlag, Heidelberg, Germany, 1984.

5. Kent, E. W., *The Brains of Men and Machines,* McGraw-Hill Publishing Co., New York. New York, 1981.

6. Barnacle, H. E., *Mechanics of Automobiles,* The MacMillan Company, New York, New York, 1964.

7. Saunders, G. H., *Dynamics of Helicopter Flight,* John Wiley & Sons, Inc., New York, New York, 1975.

8. Fenton, R. E. and Olson, K. W., "The Electric Highway," *IEEE Spectrum,* Vol. 6, No. 7, July 1969, pp. 60-66.

9. Bender, J. G., Fenton, R. E., and Olson, K. W., "An Experimental Study of Vehicle Automatic Longitudinal Control," *IEEE Transactions on Vehicular Technology,* Vol. VT-20, No. 4, November 1971, pp. 114-123.

10. Fenton, R. E., "Automatic Vehicle Guidance and Control - A State of the Art Survey," *IEEE Transactions on Vehicular Technology,* Vol. VT-19, No. 1, February 1970, pp. 153-161.

11. Fenton, R. E., Melocik, G. C., and Olson, K. W., "On the Steering of Automated Vehicles: Theory and Experiment," *IEEE Transactions on Vehicular Technology,* Vol. AC-21, No. 3, June 1976, pp. 306-315.

12. Takasaki, G. M. and Fenton, R. E., "On the Identification of Vehicle Longitudinal Dynamics," *IEEE Transactions on Automatic Control*, Vol. AC-22, No. 4, August 1977, pp. 610-615.

13. Fenton, R. E., and Chu, P. M., "On Vehicle Automatic Longitudinal Control," *Transportation Science*, Vol. 11, No. 1, February 1977, pp. 73-91.

14. Fenton, R. E. and Selim, I., "On the Optimal Design of a Vehicle Lateral Controller," *Proc. of 34th IEEE Vehicular Technology Conference*, Pittsburg, Pennsylvania, May 21-23, 1984.

15. Cormier, W. H. and Fenton, R. E., "On the Steering of Automated Vehicles - A Velocity-Adaptive Controller," *IEEE Transactions on Vehicular Technology*, Vol. VT-29, No. 4, November 1980, pp. 375-385.

16. Hauksdottir, A. S. and Fenton, R. E., "On the Design of a Vehicle Longitudinal Controller," *IEEE Transactions on Vehicular Technology,* Vol. VT-34, No. 4, November 1985, pp. 182-187.

17. El-Deen, Y. H. and Seireg, A., "Mechatronics for Cars: Integrating Machines and Electronics to Prevent Skidding on Icy Roads," *Computers in Mechanical Engineering*, Vol. 5, No. 4, January 1987, pp. 10-22.

18. Muller, D. T., *Automated Guided Vehicles*, IFS (Publications) Ltd., New York, New York, 1983.

19. Waldron, K. J. and McGhee, R. B., "The Adaptive Suspension Vehicle," *IEEE Control Systems Magazine*, Vol. 6, No. 6, December 1986, pp. 7-12.

20. Nitao, J. J., and Parodi, A. M., "A Real-Time Reflexive Pilot for an Autonomous Land Vehicle," *IEEE Control Systems Magazine*, Vol. 6, No. 1, February 1986, pp. 14-23.

21. Lowrie, J., *The Autonomous Land Vehicle 1st Quarterly Report*, Martin Marietta Denver Aerospace, Denver, Colorado, December 1985.

22. Shapiro, L. G. "The Role of AI in Computer Vision," *Proc. of the 2nd IEEE Conf. on Artificial Intelligence Applications*, Maimi Beach, Florida, December 11-13, 1985.

23. Tenenbaum, J. M., "On Locating Objects by Their Distinguishing Features," *Computer Graphics and Image Processing*, Vol. 2, 1973.

24. Brooks, R. A., *Symbolic Reasoning Among 3-D Models and 2-D Images*, Stanford Artificial Intelligence Laboratory Memo AIM-343, Stanford University, Stanford, California, June 1981.

25. Dongarra, J. J., "A Survey of High-Performance Computers," *IEEE Compcon*, March 1986, pp. 8-11.

26. Hillis, W. D., *The Connection Machine*, The MIT Press, Cambridge, Massachusetts, 1985.

27. Cuadrado, J. L. and Cuadrado, C. Y., "AI in Computer Vision," *Byte*, Vol. 11, No. 1, January 1986, pp. 237-258.

28. Minsky, M., *A Framework For Representing Knowledge*, MIT AI Memo, Massachusetts Institute of Technology. Cambridge, Massachusetts, 1974.

29. Poulos, D. D., *Range Image Processing For Local Navigation of an Automonous Land Vehicle*, M. S. Thesis, Naval Postgraduate School, Monterey, California, September, 1986.

30. McGhee, R. B., Zyda, M. J., and Tan, C. H., *A Simulated Study of an Autonomous Steering System for On-Road Operation of Automotive Vehicles*, Rep. No. NPS52-86-025, Naval Postgraduate School, Monterey, California, December, 1986.

31. McGhee, R. B., "An Approach to Computer Coordination of Motion for Energy-Efficient Walking Machines," *Bull. Mech. Engr. Lab.*, No. 43, pp. 1-21, Ibaraki. Japan. 1986.

32. Anon., *IRIS User's Guide*, Volume II-Graphics Reference, Version 3.0, Silicon Graphics ,Inc., Mountain View, California, 1986.

33. Anon., *The UNIX System User's Manual,* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

34. Manley, J. E., *A Multimedia Computer Conferencing System*, M. S. Thesis, Naval Postgraduate School, Monterey, California, 1986.

35. Stallings, W., *Local Networks, An Introduction*, Macmillan Publishing Company, Inc., New York, New York, 1984.

36. Bourne, S. R., *The UNIX System*, Addison-Wesley Publishing Company, Menlo Park, California, 1983.

37. Giordano, F. R. and Weir, M. D., *A First Course in Mathematical Modeling*, Brooks/Cole Publishing Company, Monterey, California, 1983.

APPENDIX A

SOURCE CODE FOR THE NAVIGATOR'S DISPLAY

```
/*********************************************

filename: NAVIGATE.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "gl.h"
#include "device.h"
#include "const.h"
#include "vars.h"
#include "stdio.h"
#include "math.h"
#include "time.h"
#include <sys/types.h>
#include <sys/times.h>
#include "sys/signal.h"

main()
   {

                    /*  variables to control the car  */

   int lightcolor     = REDLIGHT;   /* color of signal light  */
   int command;                     /* read/write variable */
   int condition       = 0;         /* signal received from simulation */
   int status          = 0;         /* read/write variable  */
   int car;                         /* socket number of local system  */
   int nbyte;                       /* read/write result */
   int mode             = 0;        /* control mode sent to vehcle */
   int datatime        = 0;         /* time counter for recording data */
   int systemclock;                 /* contains the system clock time */
   int starttime;                   /* initial value for the system clock */
   int mousex           = 0;        /* heading info from mouse  */
```

```
int thousand, hundred, ten, unit;
int statussize, distancesize, cxsize, cysize, commandsize,
    controlsize, velocitysize, carstatsize;
int timedata[1000];                /* used to record real time */

long controlsignal  = 0;       /* heading and brake info sent to car */
long carstat        = 0;       /* cmdvelocity and braking info from
                                        cardriver */
long seedval;                  /* used to seed random number generator */
long float drand48();              /* the random number generator  */
void srand();                  /* used to seed random number generator  */
float gaussiangenerator();         /* random number generator  */

char tempstr[30], thouc[2], hundc[2], tenc[2], unitc[2];

float rdistance     = 0.0;   /* distance the vehicle is down the road */
float a_max         = 1.8;
float temp, tempval;

Boolean notdone     = TRUE;        /* used to control display loop  */
Boolean recorddata = FALSE;       /* used to record data           */

extern long time();              /* System clock */
long clocktime;                  /* For clock value */
char *clockc;

struct tms mytime;                /* Place to put the time structure */

                /* function that connects server to client */
int connect_client();

statussize = sizeof(status);
distancesize = sizeof(rdistance);
cxsize = sizeof(cx);
cysize = sizeof(cy);
commandsize = sizeof(command);
controlsize = sizeof(controlsignal);
velocitysize = sizeof(velocity);
carstatsize = sizeof(carstat);

                /*  open up the net path to machine npscs-iris1 */
car = connect_client("npscs-iris2", 5);
```

```
loadintarray();     /* load the vision data */

ginit();                        /* initialize the IRIS  */

doublebuffer();                 /* use doublebuffer mode  */

gconfig();                      /* use the above settings */

cursoff();                      /* set the cursor off  */

qdevice(KEYBD);                   /* check input fm keyboard  */

setvaluator(MOUSEX, 250, 0, 500);
setvaluator(MOUSEY, -10, -10, 400);
noise(MOUSEX, 10);
noise(MOUSEY, 10);

color(BLACK);                     /* clear the buffers  */
clear();
swapbuffers();
clear();
swapbuffers();

makethemapview(&mapobj);
makethegauges(&gauges);

welcome();                        /* display the welcome panel */

seedval = time((long*)0);       /* seed the random number generator */
srand48(seedval);
                                  /* generate random numbers  */
for (stopcount = 0; stopcount < 10; ++stopcount)
     {
     ksubi[stopcount] = gaussiangenerator();
     ksubf[stopcount] = gaussiangenerator();
     }
stopcount = 0:

for (cyclecount = 0; cyclecount < 150; ++cyclecount)
     {
     ksube[cyclecount] = ((drand48()) * 0.01) + 0.01;
     ksubn[cyclecount] = ((drand48()) * 0.01) + 0.01;
     }
```

```
cyclecount = 0;

while(notdone)
   {
    nbyte = write(car, &command, commandsize);
    nbyte = write(car, &controlsignal, controlsize);
    nbyte = read(car, &status, statussize);
    nbyte = read(car, &velocity, velocitysize);
    notdone = status/100;
    lightcolor = ((status - (notdone * 100))/10);
    condition = status - (notdone * 100) - (lightcolor * 10);
 switch(condition)
    {
    case ASteerNSp :
    case DrSteerNavSp:
       mousespeedinput(&cmdvelocity, &distance, &eye, &numsights,
                       &lastremembered, &brakeposition, &accel_brake);
       break;
    case NavManual:
       mousespeedinput(&cmdvelocity, &distance, &eye, &numsights.
                       &lastremembered, &brakeposition, &accel_brake);
       mousex = getvaluator(MOUSEX);
       break;
    case NavSteerDrSp:
       mousex = getvaluator(MOUSEX);
       if (distance % LAPDIST > vision[eye][LOCATION])
          {
          if (vision[eye][OBJECT] == SPEEDLIMIT)
             lastremembered = vision[eye][SPEED];
          if (eye < numsights) eye = eye + 1;
          }
       nbyte = read(car, &carstat, carstatsize);
       accel_brake = -carstat/1000;
       cmdvelocity = (int)((carstat%1000) * MPS_TO_KMPH);
       break;

    case ASteerDrSp: if (distance % LAPDIST > vision[eye][LOCATION])
          {
          if (vision[eye][OBJECT] == SPEEDLIMIT)
             lastremembered = vision[eye][SPEED];
          if (eye < numsights) eye = eye + 1;
          }
       nbyte = read(car, &carstat, carstatsize);
```

```
    accel_brake = -carstat/1000;
    cmdvelocity = (int)((carstat%1000) * MPS_TO_KMPH);
    break;
case DrManual:if (distance % LAPDIST > vision[eye][LOCATION])
        {
        if (vision[eye][OBJECT] == SPEEDLIMIT)
            lastremembered = vision[eye][SPEED];
        if (eye < numsights) eye = eye + 1;
        }
    nbyte = read(car, &carstat, carstatsize);
    accel_brake = -carstat/1000;
    cmdvelocity = (int)((carstat%1000) * MPS_TO_KMPH);
    break;
case CruiseNavSteer: mousex = getvaluator(MOUSEX);
case AUTOPILOT:
case CruiseDrSteer:
    cruisecontrol(systemclock, a_max, lightcolor);
    cmdvelocity = cruisevelocity;
    break;
}  /* switch(condition)    */

            /* reset vision array after every lap */
if ((distance + 1) % (LAPDIST - 1) == 0)
    eye = 0;

nbyte = read(car, &cx, cxsize);
nbyte = read(car, &cy, cysize);
nbyte = read(car, &rdistance, distancesize);

distance = rdistance;
thousand = distance/1000;
hundred = (distance - (thousand * 1000))/100;
ten = (distance - (thousand * 1000) - (hundred *100))/10;
unit = distance - (thousand * 1000) - (hundred * 100) - (ten * 10);

clocktime = time((long*)0);          /* record the system clock */
clockc = ctime(&clocktime);
systemclock = times(&mytime);
sprintf(thouc, "%d", thousand);
sprintf(hundc, "%d", hundred);
sprintf(tenc, "%d", ten);
sprintf(unitc, "%d", unit);
```

```
editobj(mapobj);
if (lightcolor == GREENLIGHT)
    {
    objreplace(greenlighttag);
    color(GREEN);
    objreplace(redlighttag);
    color(DIMRED);
    }
if (lightcolor == YELLOWLIGHT)
    {
    objreplace(yellowlighttag);
    color(YELLOW);
    objreplace(greenlighttag);
    color(DIMGREEN);
    }
if (lightcolor == REDLIGHT)
    {
    objreplace(redlighttag);
    color(RED);
    objreplace(yellowlighttag);
    color(DIMYELLOW);
    }
objreplace(cartag);
move2(cx-5.0,-cy);
draw2(cx+5.0,-cy);
move2(cx, -cy-5.0);
draw2(cx, -cy+5.0);

            /*   display the system time    */
objreplace(timetag);
charstr(clockc);

if (recorddata)
    {
    sprintf(tempstr, "%s ki %.3f kv %.3f", "RECORDING",
            ksubi[stopcount], ksubf[stopcount]):
    objreplace(testtag);
    charstr(tempstr);
    }
  else
    {
    sprintf(tempstr, "ki %.3f kf %.3f",
            ksubi[stopcount], ksubf[stopcount]):
```

```
        objreplace(testtag);
        charstr(tempstr);
        }
    closeobj();
    callobj(mapobj);

    /* EDIT GUAGES */

    sprintf(tempstr, "%d", (distance/LAPDIST) + 1);

    editobj(gauges);
    objreplace(cmdvelocitytag);
    rectfi(CMDX, CMDY, CMDX + 50, CMDY + 2 * cmdvelocity);
    objreplace(carvelocitytag);
    rectfi(CARVELX, CARVELY, CARVELX + 50,
(int)(CARVELY + 2 * velocity));
    objreplace(brakepositiontag);
    rectfi(BRAKEX, BRAKEY, BRAKEX + 50, BRAKEY - accel_brake);
    objreplace(laptag);
    charstr(tempstr);
    objreplace(thoutag);
    charstr(thouc);
    objreplace(hundredtag);
    charstr(hundc);
    objreplace(tentag);
    charstr(tenc);
    objreplace(unittag);
    charstr(unitc);

    switch(condition)
        {
        case AUTOPILOT:  objreplace(modeinserttag);
                         charstr("Q");
                         objreplace(modetag);
                         charstr("AutoPilot");
                         cmov2i(LAPX - 26, LAPY - 72);
                         charstr("   ");
                         closeobj();
                         break;

        case CruiseDrSteer:objreplace(modeinserttag);
                         charstr("C");
                         objreplace(modetag);
```

```
                    charstr("Cruise Control");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Driver St Cont");
                    closeobj();
                    break;

case CruiseNavSteer:objreplace(modeinserttag);
                    charstr("R");
                    objreplace(modetag):
                    charstr("Cruise Control");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Nav St Cont");
                    closeobj();
                    break;

case ASteerDrSp: objreplace(modeinserttag);
                    charstr("S");
                    objreplace(modetag);
                    charstr("AutoSteer");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Driver Sp Cont");
                    closeobj();
                    break;

case ASteerNSp:   objreplace(modeinserttag);
                    charstr("A");
                    objreplace(modetag);
                    charstr("AutoSteer");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Nav Sp Cont");
                    closeobj();
                    break;

case DrManual:    objreplace(modeinserttag);
                    charstr("D");
                    objreplace(modetag);
                    charstr("Driver Manual");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("No Nav Cont");
                    closeobj();
                    break;

case NavManual:     objreplace(modeinserttag);
```

```
                    charstr("W");
                    objreplace(modetag);
                    charstr("Nav Manual");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Total Control");
                    closeobj();
                    break;

    case DrSteerNavSp: objreplace(modeinserttag);
                    charstr("X");
                    objreplace(modetag);
                    charstr("Driver Steer");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Nav Speed");
                    closeobj();
                    break;

    case NavSteerDrSp: objreplace(modeinserttag);
                    charstr("F");
                    objreplace(modetag);
                    charstr("Nav Steer");
                    cmov2i(LAPX - 26, LAPY - 72);
                    charstr("Driver Speed");
                    closeobj();
                    break;
    }

callobj(gauges);

checkkeybd(&notdone, &recorddata, &mode, &condition,
            &stopcount, velocity);

command = (mode * 1000) + cmdvelocity;
controlsignal = (mousex * 1000) - accel_brake;

if (recorddata)
    {
    data[datatime][DISTANCE] = rdistance;
    data[datatime][VELOCITY] = velocity;
    cmdvel[datatime] = cmdvelocity;
    brakedata[datatime] = -accel_brake;
    timedata[datatime] = systemclock - starttime;
    if (condition == CruiseNavSteer | condition == AUTOPILOT |
```

```
                condition == CruiseDrSteer)
                {
                est_i[datatime] = ksubi[stopcount];
                est_f[datatime] = ksubf[stopcount];
                }
              else
                {
                est_i[datatime] = 0.0;
                est_f[datatime] = 0.0;
                }
            ++datatime;
            }
          else  starttime = times(&mytime);

         swapbuffers();
        }
        close(car);
                        /* writes data to a file */
savedata(data. datatime. brakedata, timedata, cmdvel);
color(BLACK);
clear();
swapbuffers();
clear();
swapbuffers();
finish();
gexit();
}


/*********************************************

filename: CRUISE.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"
#include <math.h>

cruisecontrol(systemclock, a_max, lightcolor)
int systemclock, lightcolor;
```

```
float a_max;
   {
   float temp;
          switch(vision[eye][OBJECT])
          {
          case SPEEDLIMIT:
                if ((distance % LAPDIST) >= vision[eye][LOCATION])
                   {
                    cruisevelocity = vision[eye][SPEED];
                    lastremembered = vision[eye][SPEED];
                    if (eye < numsights) eye = eye + 1;
                   }
                else cruisevelocity = lastremembered;
                break;

           case STOPSIGN:
                PLANNED_DIST = vision[eye][LOCATION] - distance;
                if (PLANNED_DIST > 0)
                   {
                   temp = 2 * a_max * PLANNED_DIST;
                   PLANNED_VELOCITY = 3.6 * (sqrt(temp));
                   }
                else PLANNED_VELOCITY = 0.0;
                if (velocity > PLANNED_VELOCITY) stopping = TRUE;
                if (velocity == 0.0) clearintersection(systemclock);
                else if (stopping) stopcar();
                break;

           case SIGNALLIGHT:
                PLANNED_DIST = vision[eye][LOCATION] - distance;
                if (PLANNED_DIST > 0)
                   {
                   temp = 2 * a_max * PLANNED_DIST;
                   PLANNED_VELOCITY = 3.6 * (sqrt(temp));
                   }
                else PLANNED_VELOCITY = 0.0;
                if (velocity > PLANNED_VELOCITY) stopping = TRUE;
                if (stopping) signallight(lightcolor, systemclock);
                 break;

           default:
                break;
            } /* end switch */
```

```
}

/*********************************************

filename: BRAKE.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"

stopcar()

{
float pervel, perdist, temp, temp1;
float VELOCITY_GAIN_FACTOR;
float POSITION_GAIN_FACTOR;
float ALPHA = 0.60;

VELOCITY_GAIN_FACTOR = 4.289;
POSITION_GAIN_FACTOR = 4.840;

/*                      Used to calibrate system
ksubi[stopcount] = 1.0;
ksubf[stopcount] = 1.0;
ksube[cyclecount] = 0.0;
ksubn[cyclecount] = 0.0;
*/

                    /* what does the driver perceive */
perdist = (ksubi[stopcount] + ksube[cyclecount]) * (PLANNED_DIST);

pervel = (ksubf[stopcount] + ksubn[cyclecount]) * velocity;

if (PLANNED_DIST < 0) PLANNED_DIST = 0;

accel_brake = (int)(-(POSITION_GAIN_FACTOR*(PLANNED_DIST-perdist)) +
                    (VELOCITY_GAIN_FACTOR *
                    (PLANNED_VELOCITY - pervel)) - ALPHA * ON);
cruisevelocity = 0;
if (accel_brake > 0) accel_brake = 0;
```

```
if (accel_brake < -200) accel_brake = -200;
++cyclecount;

temp = ksubi[stopcount] + ksube[cyclecount - 1];
temp1 = ksubf[stopcount] + ksubn[cyclecount - 1];
}


/*********************************************

filename: SIGNAL.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"

signallight(lightcolor, systemclock)

int lightcolor, systemclock;
{
switch(lightcolor)
      {
      case GREENLIGHT :

            brakeposition = OFF;
            accel_brake = OFF;
            cruisevelocity = lastremembered;
            if (eye < numsights) eye = eye + 1;
            firstcall = TRUE;
            stopping = FALSE;
            ++stopcount;
            cyclecount = 0;
            break:
       case YELLOWLIGHT :

            stopcar();
            break;

       case REDLIGHT :

            stopcar();
```

```
            break;
        }
}

/*********************************************

filename: CLEAR.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"

clearintersection(systemclock)
int systemclock;

{
if (firstcall)
    {
    savedtime = systemclock;
    firstcall = FALSE;
    }
    else if (systemclock - savedtime > 120)
        {
        brakeposition = OFF;
        accel_brake = OFF;
        firstcall = TRUE;
        stopping = FALSE;
        if (eye < numsights) eye = eye + 1;
        cruisevelocity = lastremembered;
        ++stopcount;
        cyclecount = 0;
        }
}

/*********************************************

filename: MAPVIEW.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/
```

```
#include "gl.h"
#include "vars.ext.h"
#include "stdio.h"
#include "const.h"
makethemapview(mapobj)
Object *mapobj;
{
int i;                          /* loop control  */

mapcolor(DIMGREEN, 0, 110, 0);
mapcolor(DIMYELLOW, 170, 170, 0);
mapcolor(DIMRED. 110, 0, 0);
mapcolor(LIGHTBLUE, 0, 255, 255);

*mapobj = genobj();
makeobj(*mapobj);

pushmatrix();                   /* save the stack  */
pushviewport();                 /* save the viewpoet  */
viewport(0, 767, 0, 767);      /* reset the viewport  */
                                /* coordinate system set to match the
                                   coordinates fo the road.  */
ortho2(-108.0, 660.0, -184.0, 584.0);
                                /* fill the background  */
color(LIGHTBLUE);
rectf(-108.0, -184.0, 660.0, 584.0);
color(BLACK);

     /* Draw the straight roadlengths  */
color(BLACK);
rectf(-12.0, 0.0, 4.0, 400.0);          /* Length #1  */
rectf(76.0, 472.0, 476.0. 488.0);       /* Length #2  */
rectf(548.0. 0.0, 564.0. 400.0);        /* Length #3  */
rectf(72.0. -88.0. 476.0, -72.0);       /* Length #4  */

      /* Draw the corners  */
color(BLACK);
arcfi(76, 400, 88, 900, 1800);          /* Draw first arc  */
color(LIGHTBLUE);
arcfi(76, 400, 71, 900, 1800);          /* Backfill the offroad area  */

color(BLACK);
arcf(476.0, 400.0, 88.0, 0, 900);       /* Draw second arc  */
```

```
color(LIGHTBLUE);
arcf(476.0, 400.0, 71.0, 0, 900);      /* Backfill the offroad area  */

color(BLACK);
arcf(476.0, 0.0, 88.0, 2700, 0);       /* Draw third arc  */
color(LIGHTBLUE);
arcf(476.0, 0.0, 71.0, 2700, 0);       /* Backfill the offroad area  */

color(BLACK);
arcf(76.0, 0.0, 88.0, 1800, 2700);     /* Draw fourth arc  */
color(LIGHTBLUE);
arcf(76.0, 0.0, 71.0, 1800, 2700);     /* Backfill the offroad area  */

color(BLACK);
rectf(-50.0, 92.0, 595.0, 108.0);      /* draw the first crossroad  */
rectf(-50.0, 292.0, 595.0, 308.0);     /* draw the second crossroad  */

cmov2i(16, 5);                             /* mark the start  */
charstr("START");

stopsign(32.0, 113.0, 1);              /* add the stopsigns  */
stopsign(519.0, 113.0, 3);
stopsign(519.0, 316.0, 2);

color(BLACK);                              /* make a signallight  */
rectfi(12, 314, 43, 394);

redlighttag = gentag();
maketag(redlighttag):
color(DIMRED);
circfi(27, 378, 10);

yellowlighttag = gentag();
maketag(yellowlighttag);
color(DIMYELLOW);
circfi(27, 353, 10);

greenlighttag = gentag();
maketag(greenlighttag);
color(DIMGREEN);
circfi(27, 328, 10);
                /* mark the car with crosshair */
color(WHITE);
```

```
linewidth(2);
cartag = gentag();
maketag(cartag);
move2(0.0,0.0);
draw2(0.0, 0.0);
move2(0.0,0.0);
draw2(0.0, 0.0);
linewidth(1);
                    /* display the system time  */
cmov2i(-50, 560);
color(RED);
timetag = gentag();
maketag(timetag);
charstr("       ");
                    /* indicate if recording data */
cmov2i(250. 560);
testtag = gentag();
maketag(testtag);
charstr(" "):
                    /* return the state of the machine.  */
popviewport();
popmatrix();
closeobj();

}

/***********************************************
                    stopsign()
***********************************************/
stopsign(xpos, ypos, num)
float xpos, ypos;
int num;
{
char tempstr[5];
Coord vertice[10][2];

float width = 42.0;
float temp = 5 * width/24;
float temp1 = 7 * width/24;

vertice[0][0] = xpos;
vertice[0][1] = ypos;
```

```
vertice[1][0] = xpos + temp;
vertice[1][1] = ypos;

vertice[2][0] = xpos + width/2;
vertice[2][1] = ypos + temp1;

vertice[3][0] = xpos + width/2;
vertice[3][1] = temp1 + (2 * temp) + ypos;

vertice[4][0] = xpos + temp;
vertice[4][1] = ypos + width;

vertice[5][0] = xpos - temp;
vertice[5][1] = ypos + width;

vertice[6][0] = xpos - width/2;
vertice[6][1] = temp1 + (2 * temp) + ypos;

vertice[7][0] = xpos - width/2;
vertice[7][1] = temp1 + ypos;

vertice[8][0] = xpos - temp;
vertice[8][1] = ypos;
                /* fill the sign in red  */
color(RED);
polf2(9, vertice);
                /*  outline the sign  */
color(WHITE);
poly2(9, vertice);
                /* put stop on the sign  */
cmov2(xpos - 16, ypos + 16);
charstr("STOP");

color(BLACK);
cmov2(xpos - 3.0, ypos + width + 5.0);
sprintf(tempstr. "%d", num);
charstr(tempstr);

}

/*********************************************

filename: GAUGES.C
```

```
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"

makethegauges(gauges)
Object *gauges;
{

*gauges = genobj();
makeobj(*gauges);

pushmatrix();
pushviewport();

viewport(768, 1023. 0, 767);
ortho2(0.0, 255.0, 0.0, 767.0);
                    /* fill the backcolor  */
color(WHITE);
rectf(0.0, 0.0, 255.0, 767.0);
                    /* label the gauge */
color(BLACK);
cmov2i(32, 730);
charstr("COMMAND");
cmov2i(159, 730);
charstr("VEHICLE");
cmov2i(40, 714);
charstr("SPEED");
cmov2i(155, 714);
charstr("VELOCITY");

                    /* draw the command speed gauge */
color(GREEN):
cmdvelocitytag = gentag();
maketag(cmdvelocitytag);
rectfi(CMDX, CMDY, CMDX + 50, CMDY);
scalegauge(CMDX, CMDY);
                    /* draw the car velocity gauge */
color(GREEN);
carvelocitytag = gentag();
```

```
maketag(carvelocitytag);
rectfi(CARVELX, CARVELY, CARVELX + 50, CARVELY);
scalegauge(CARVELX, CARVELY);
                    /* draw the brake pressure gauge */
color(BLACK);
cmov2i(41, 470);
charstr("BRAKE");
cmov2i(28, 454);
charstr("PRESSURE");

color(RED);
brakepositiontag = gentag();
maketag(brakepositiontag);
rectfi(BRAKEX, BRAKEY, BRAKEX + 50, BRAKEY);
scalegauge(BRAKEX, BRAKEY);
                    /* draw the distance indicator */
color(BLACK);
cmov2i(177, 470);
charstr("LAP");
cmov2i(155, 454);
charstr("DISTANCE");
                    /* outline the box */
linewidth(2);
recti(LAPX, LAPY, LAPX + 80, LAPY + 60);
linewidth(1);

cmov2i(LAPX + 35, LAPY + 35);
laptag = gentag();
maketag(laptag);
charstr("1");

cmov2i(LAPX + 9, LAPY + 16);
thoutag = gentag();
maketag(thoutag);
charstr("0");

cmov2i(LAPX + 27, LAPY + 16);
hundredtag = gentag();
maketag(hundredtag);
charstr("0");

cmov2i(LAPX + 45, LAPY + 16);
tentag = gentag();
```

```
maketag(tentag);
charstr("0");

cmov2i(LAPX + 63, LAPY + 16);
unittag = gentag();
maketag(unittag);
charstr("0");
                    /* set up the mode indicator  */
cmov2i(LAPX - 8, LAPY - 40);
charstr("Mode: ");
color(RED);
modeinserttag = gentag();
maketag(modeinserttag);
charstr("D");

cmov2i(LAPX - 26, LAPY - 56);
color(RED);
modetag = gentag();
maketag(modetag);
charstr("Driver Manual");
cmov2i(LAPX - 26, LAPY - 72);
charstr("No Nav Cont");

color(BLACK);
                    /* Display help information */
cmov2i(168, 290);
charstr("MOUSE");

cmov2i(155, 274);
charstr("Steering");

cmov2i(155, 239);
charstr("Brakes");

/* draw a direction arrow for left/right and up/down */
linewidth(2);
move2i(155, 261);
draw2i(225, 261);

move2i(165, 266);
draw2i(155, 261);

move2i(165, 256);
```

```
draw2i(155, 261);

move2i(215, 266);
draw2i(225, 261);

move2i(215, 256);
draw2i(225, 261);


move2i(225, 250);
draw2i(225, 205);

move2i(220, 215);
draw2i(225, 205);

move2i(230, 215);
draw2i(225, 205);

move2i(220, 240);
draw2i(225, 250);

move2i(230, 240);
draw2i(225, 250);

linewidth(1):

cmov2i(65, 214);
charstr("KEYBD CONTROLS");

cmov2i(30, 194);
charstr("Q: AutoPilot");

cmov2i(30, 174);
charstr("C: Cruise, Dr Steer");

cmov2i(30, 154):
charstr("R: Cruise, Nav Steer");

cmov2i(30, 134);
charstr("S: AutoSt, Dr Speed");

cmov2i(30, 114);
charstr("A: AutoSt, Nav Speed");
```

```
cmov2i(30, 94);
charstr("D: Driver Control");

cmov2i(30, 74);
charstr("W: Nav Control");

cmov2i(30, 54);
charstr("X: Dr Steer, Nav Sp");

cmov2i(30, 34);
charstr("F: Nav Steer, Dr Sp");

cmov2i(30, 14);
charstr("E: Exit");

popviewport();
popmatrix();
closeobj();
}

/*********************************************
                scalegauge();
*********************************************/

scalegauge(basex, basey)
int basex, basey;
{
char tempstr[10];
int i;
                    /* outline the gauge */
linewidth(2);
color(BLACK);
recti(basex, basey, basex + 50, basey + 200);
linewidth(1);
                    /* calibrate the gauge */
for (i = 10; i < 100; i = i + 10)
    {
    move2i(basex, basey + 2 * i);
    draw2i(basex + 13, basey + 2 * i);
    move2i(basex + 37, basey + 2 * i);
    draw2i(basex + 50, basey + 2 * i);
    cmov2i(basex + 16, basey + (2 * i) - 4);
    sprintf(tempstr, "%d", i);
```

```
    charstr(tempstr);
    }
}    /*  scalegauge()  */

/*******************************************

filename: MOUSE.C
author: Michael J. Dolezal
date: May 20, 1987

*******************************************/

#include "const.h"
#include "vars.ext.h"

mousespeedinput(cmdvelocity, distance, eye, numsights, lastremembered,
                brakeposition, accel_brake)
int *cmdvelocity, *distance, *eye, *numsights, *lastremembered,
    *brakeposition, *accel_brake;
{
        if (getbutton(MOUSE1))
            {
            *cmdvelocity = *cmdvelocity + SPEEDINC;
            if (*cmdvelocity > 100)
                {
                *cmdvelocity = 100;
                }
             }
        if (getbutton(MOUSE2))          /* decrease speed */
            {
            *cmdvelocity = *cmdvelocity - SPEEDINC;
            if (*cmdvelocity < 0) *cmdvelocity = 0;
            }
        if (*distance % LAPDIST > vision[*eye][LOCATION])
            {
            if (vision[*eye][OBJECT] == SPEEDLIMIT)
                *lastremembered = vision[*eye][SPEED];
            if (*eye < *numsights) *eye = *eye + 1;
            }
        *brakeposition = (getvaluator(MOUSEY))/2;
        if (*brakeposition < 0) *brakeposition = 0;
        *accel_brake = -(*brakeposition);
}
```

```
/*********************************************

filename: NETV.C
author: James Manley
modified by: Michael Zyda
date: April 29, 1987

*********************************************/

/*

This is file netV.c
as modified by M. Zyda, 29 April 1987

This segment, when linked into a program on a computer with a UNIX 4.2 BSD
operating system, will allow the program to communicate with programs
executing on other computer systems over an Internet network.

*/

#define TRUE 1

                    /* include files for UNIX 4.2 BSD */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <bsd/netdb.h>

/*********************************************************

The connect_server(remote_client_name, port_number) function performs
the actions required to connect a server system to a remote client system

*********************************************************/

int connect_server(remote_client_name, port_number)

                    /* name and port number of the remote client system */

    char remote_client_name[];
    int  port_number;
```

```
{
                /* pointer to the remote client system's name */

char *ptr_client_name;

                /* local socket number */

int local_server_socket;

                /* function that opens a socket */

int socket();

                /* function that accepts a connection from a remote client
                   socket */

int accept();

                /* socket number of the remote client system */

int remote_client_socket;

                /* protocol and address data structure specified for the
                   socket */

static struct sockaddr_in address  = { AF_INET };

                /* address of the remote client system */

long remote_client_address;
                /* port number of the remote client system */

short remote_client_port;

                /* size of the address data structure of the remote client
                   system */
int address_size;
                /* begin the process of attempting to connect to the
                   remote client system */

                /* get a pointer to the remote client system's name */

ptr_client_name = &remote_client_name[0];
```

```
                /* convert the remote client system name to its address */

remote_client_address = (long)gethostbyname(&ptr_client_name);

                /* initialize the remote client port number above the
                   system reserved ports */

remote_client_port = IPPORT_RESERVED + 1;

                /* add the remote client port number to the number of
                   reserved ports */

remote_client_port = remote_client_port + port_number;

                /* initialize the remote client socket number to an
                   invalid value */

remote_client_socket = -1;
                /* remote client system address family (Internet in this
                   case) */

address.sin_family = AF_INET ;

                /* place the remote client port number into the address
                   data structure */

address.sin_port = remote_client_port;

                /* put the port number in network byte order */

address.sin_port = htons(address.sin_port);

                /* place the remote client system's address in the address
                   data structure */

address.sin_addr.s_addr = remote_client_address:

                /* find number of bytes in the remote client address */

address_size = sizeof(remote_client_address);

                /* attempt to open a local socket */
```

```
local_server_socket = socket(AF_INET,SOCK_STREAM,0);
if(local_server_socket < 0)
    {

                /* the server couldn't open a local socket */

    perror("Server couldn't open a local socket:");
    }
    else
    {
    if(bind(local_server_socket, (caddr_t)&address,
            sizeof(address)) < 0)
        {
        perror("Server couldn't bind address to local socket: ");
        }

                /* set the maximum number of remote client systems to
                   be connected to */

    listen(local_server_socket,5);
    printf("Server waiting to connect to %s\n",remote_client_name);

                /* attempt to accept a connection */

    remote_client_socket = accept(local_server_socket, &address,
                                  &address_size);

    if (remote_client_socket < 0)
        {

                /* an error occurred in the server attempting to
                   accept a connection from remote client system */

        perror("Server couldn't accept a connection
                from the  remote client system :");
        close(local_server_socket);
        }
    } /* end else local_server_socket >= 0 */
                /* return the socket number of the remote client system */

return(remote_client_socket);

} /* end of function connect_server()  */
```

```
/*************************************************************

The connect_client(remote_server_name port_number) function performs
all the actions required to connect a client system to a remote server
system

*************************************************************/

int connect_client(remote_server_name, port_number)

                    /* name and port number of the remote server system */

    char remote_server_name[];
    int  port_number;

    {

                    /* pointer to the name of the remote server system */

    char *ptr_server_name;

                    /* local socket number */

    int local_client_socket;

                    /* function that opens a socket */

    int socket();

                    /* function that connects local socket to remote server
                       socket */

    int connect();

                    /* the socket number on the remote server system  */

    int remote_server_socket;

                    /* the protocol and address data structure specified for
                       the socket */

    static struct sockaddr_in address = { AF_INET };
```

```
                /* address of the remote server system */

struct hostent *remote_server_address;

                /* port number of remote system */

short remote_server_port;

                /* begin the process of attempting to connect to the
                   remote server system */

                /* establish ptr to the remote server name */

ptr_server_name = &remote_server_name[0];
                /* convert the name of the remote server system to an
                   address */
remote_server_address = gethostbyname(ptr_server_name);

                /* clear out the address structure */

bzero((char *)&address, sizeof(address));

                /* copy the remote server address structure into the
                   address structure */

bcopy(remote_server_address->h_addr,

      (char *)&address.sin_addr,

      remote_server_address->h_length);

                /* initialize remote server port number above the system
                   reserved ports */

remote_server_port = IPPORT_RESERVED + 1;

                /* add the user's remote server port number to the number
                   of reserved ports */

remote_server_port = remote_server_port + port_number;

                /*  remote server system address family(Internet in this
                    case) */
```

```
address.sin_family = AF_INET;

                /* place the remote server port number into the address
                   structure */

address.sin_port = remote_server_port;

                /* put the port number in network byte order */

address.sin_port = htons(address.sin_port);

                /* attempt to obtain a local socket */

local_client_socket = socket(AF_INET, SOCK_STREAM, 0);

if(local_client_socket < 0)
   {

                /* the client couldn't open a local socket */

   perror("Client couldn't open a local socket: ");
   }
else
   {

                /* place Internet address family type in address
                   structure */

   address.sin_family = AF_INET;

                /* attempt to connect local client socket to remote
                   server socket */
   remote_server_socket =

      connect(local_client_socket, (caddr_t)&address,

                sizeof(address));

   if(remote_server_socket < 0)
      {

                /* an error occurred in attempting to connect to
                   the remote server socket */
```

```
        perror("Client couldn't connect to the remote server socket: ");
        close(local_client_socket);
        }
    else
        {

                /* successfully connected to the remote server
                   system */

        printf("Connection established with %s.\n",remote_server_name);
        }

    } /* end else socket >= 0 */

                /* return the socket number of the local client system */

  return(local_client_socket);

  } /* end of funcition connect_client() */

/*****************************************************

function write_immediate(socket_number,buffer,nbytes) writes to the
network connection by forking a child process which actually performs the
write operation.

*****************************************************/

int write_immediate(socket_number, buffer, nbytes)
                /* socket number to be written */

  int socket_number;

                /* buffer of bytes to be written */

  char buffer[];

                /* the number of bytes to be written */

  long nbytes;

  {
```

```
                /* function which initiates a child process */

int fork();

                /* value returned from routine fork()  */

int forkval;
                /* initiate a child process which will perform the write
                   operation */

forkval = fork();

                /* determine whether a child process was successfully
                   started */

if(forkval == 0)
   {

                /* attempt to perform the write operation */

   while(write(socket_number,buffer,nbytes) != nbytes)
      {

                /* attempt to write the buffer contents */

      }

                /* terminate the child process after the buffer is
                   successfully written */

   exit(1);
   }
else
                /* an error occurred in starting the child process */

   {
   perror("Error occurred while attempting
           to fork a write-immediate process: ");

                /* return an error value to the application */

   return(-1);
   }
```

```
                    /* return a value indicating successful operation */

    return(forkval);

    } /* end of write_immediate() function */

/*********************************************

filename: CHECKKEY.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "device.h"
#include "vars.ext.h"

checkkeybd(ptnotdone, ptrecorddata, ptmode, ptcondition,
            ptstopcount, velocity)

Boolean *ptnotdone, *ptrecorddata;
int *ptmode, *ptcondition, *ptstopcount;
float velocity;
{

keypressed = NULL;

*ptmode = *ptcondition;

if (qtest())
    {
    qread(&keypressed);

        switch(keypressed)
          {
          case 'q':
          case 'Q': if (velocity > 3.0)
                        {
                        *ptmode = AUTOPILOT;
                        }
                    break;
                /* Cruise cont and driver steer */
```

```
case 'c':
case 'C': if (*ptmode == ASteerNSp || *ptmode == ASteerDrSp)
              {
              *ptmode = AUTOPILOT;
              }
              else *ptmode = CruiseDrSteer;
          break;

       /* Cruise cont and remote steer  */
case 'r':
case 'R': if (*ptmode == ASteerNSp || *ptmode == ASteerDrSp)
              {
              *ptmode = AUTOPILOT;
              }
              else *ptmode = CruiseNavSteer;
          break;

       /* Auto speed and driver speed */
case 's':
case 'S': if (velocity > 3.0)
              {
              if (*ptmode == CruiseNavSteer ||
                  *ptmode == CruiseDrSteer)
                  {
                  *ptmode = AUTOPILOT;
                  }
                  else *ptmode = ASteerDrSp;
              }
          break;

       /* Auto speed and remote steer  */
case 'a':
case 'A': if (velocity > 3.0)
              {
              if (*ptmode == CruiseNavSteer ||
                  *ptmode == CruiseDrSteer)
                 {
                 *ptmode = AUTOPILOT;
                 }
              else *ptmode = ASteerNSp;
              }
          break;
```

```
                /* All remote manual control */
        case 'w':
        case 'W': *ptmode = NavManual;
                  break;
        case 'd':
        case 'D': *ptmode = DrManual;
                  break;

        case 'x':
        case 'X': *ptmode = DrSteerNavSp;
                  break;

        case 'f':
        case 'F': *ptmode = NavSteerDrSp;
                  break;

        case 'e':
        case 'E': *ptnotdone = FALSE;
                  break:

        case 't':
        case 'T': if (*ptrecorddata) *ptrecorddata = FALSE;
                  else *ptrecorddata = TRUE;
                  break;
        }

    }
    *ptcondition = *ptmode;
}  /* checkkeybd */

/*********************************************

filename: SAVEDATA.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"
#include "stdio.h"
#include "const.h"

savedata(data, datatime, brakedata, timedata,cmdvel)
```

```
float data[][2];
int datatime;
int brakedata[];
int timedata[];
int cmdvel[];
{
FILE *fopen(), *fp;
int i;
fp = fopen("test", "w");

for (i = 0; i < datatime; ++i)
    {
    fprintf(fp, "%d %d %.3f %.3f %d %d %.3f %.3f\n", i, cmdvel[i],
            data[i][DISTANCE],
            data[i][VELOCITY], brakedata[i], timedata[i], est_i[i],
            est_f[i]);
     }
close(fp);
}

/*********************************************

filename: GENERATE.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"
#include "const.h"

float gaussiangenerator()
{
float temp;
int i;
long float randomnums[12];
long float drand48();
long float sum = 0.0;
for (i = 0; i <= 11; ++i)
     {
      randomnums[i] = (0.2 * drand48()) - 0.1;
      sum = sum + randomnums[i];
     }
```

```
temp = (sum/2.00) + 1.0;
return(temp);
}

/*********************************************

filename: LOADINTARRAY.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "vars.ext.h"

 loadintarray()
{
 if ((fp = fopen("vision.h","r")) ==NULL)
     {
     printf("Cannot see into vision.h.");
     return(-1);
     }
 else
     {
     for (eye = 0; !feof(fp); ++eye)
          {
          fscanf(fp, "%d %d %d", &vision[eye][LOCATION],
                  &vision[eye][OBJECT], &vision[eye][SPEED]);
          if (eye == 99)
              {
              printf("Vision array is full, increase size please.\n");
              break;
              }
          }
     }
 numsights = --eye;            /* number of items in vision file  */
 eye = 0;
}

*********************************************/

filename: WELCOME.C
author: Michael J. Dolezal
date: May 20, 1987
```

```
*********************************************/
#include "const.h"
#include "vars.ext.h"

static int parray[4][2] = {{275,600},{250,625},{275,625},{300,600}};
static int parray1[4][2] = {{275,475},{250,500},{275,500},{300,475}};

/*************************************************************

                W E L C O M E     D I S P L A Y

*************************************************************/

welcome()
{

    color(YELLOW);
    clear();

    color(BLUE);

    rectfi(200,625,300,700);
    rectfi(200;600,225,625);
    polf2i(4,parray);

    rectfi(325,600,425,700);
    rectfi(450,600,550,700);
    rectfi(575,600,675,700);
    polf2i(4,parray1);

    rectfi(200,475,225,500);
    rectfi(200,500,300,575);
    rectfi(325,475,425,575);
    rectfi(450,475,550,500);
    rectfi(450,500,475,575);
    rectfi(575,475,675,500);
    rectfi(575,500,600,575);
    rectfi(700,525,800,575);
    rectfi(737,475,762,525);

    color(YELLOW);

    rectfi(225,650,275,675);
```

```
    rectfi(350,625,400,675);
    rectfi(475,600,525,625);
    rectfi(475,650,525,675);
    rectfi(575,625,600,675);
    rectfi(625,625,650,675);
    rectfi(225,525,275,550);
    rectfi(350,525,400,550);
    rectfi(350,475,400,500);
    rectfi(725,550,775,575);

    color(MAGENTA);

    cmov2i(200,350);
    charstr("Welcome to the world of ROAD RALLY, a simulation");

    cmov2i(200,325);
    charstr("of a car on a road controlled by a remote driver");
    cmov2i(200,300);
    charstr("(a control program executing on another processor).");

    cmov2i(200,275);
    charstr("To exit the program press all three mouse buttons");

    cmov2i(200,250);
    charstr("at the same time. Files are now loading and the");
    cmov2i(200,225);
    charstr("demonstration will begin shortly. ");

    linewidth(5);
    cmov2i(200,150);
    charstr("Car simulation by: Tan Chiam Huat and Mike Whiting");
    cmov2i(200, 125);
    charstr("Driver simulation and networking by: Mike Dolezal");
    linewidth(1);
    swapbuffers();

} /* welcome */
/*********************************************
```

filename: CONST.H
author: Michael J. Dolezal
date: May 20, 1987

```
**********************************************/

#include "gl.h"
#include "device.h"
#include "stdio.h"
        /* defines the light conditions */
#define REDLIGHT         1
#define YELLOWLIGHT      2
#define GREENLIGHT       3
        /* used to define braking conditions */
#define OFF              0
#define ON             200
#define SPEEDLIMIT       4
#define STOPSIGN         5
#define SIGNALLIGHT      6
#define LOCATION         0
#define OBJECT           1
#define SPEED            2
#define MPS_TO_KMPH    3.6
#define DIMGREEN         9
#define DIMYELLOW       10
#define DIMRED          11
#define LIGHTBLUE       12
        /* next 6 constants are for locating guages */
#define BRAKEX          39
#define BRAKEY         250
#define CARVELX        166
#define CARVELY        510
#define CMDX            39
#define CMDY           510
#define LAPX           150
#define LAPY           390
#define DISTANCE         0
#define VELOCITY         1
#define SPEEDINC         1
#define LAPDIST       2074
#define DrManual         0
#define ASteerNSp        1
#define CruiseNavSteer   2
#define AUTOPILOT        3
#define CruiseDrSteer    4
#define ASteerDrSp       5
#define NavManual        6
```

```
#define DrSteerNavSp     7
#define NavSteerDrSp     8

/*********************************************

filename: VARS.H
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "stdio.h"

int cmdvelocity      = 0;          /* commanded vehicle velocity */
int cruisevelocity   = 0;          /* velocity read from vision file */
int distance;                      /* distance the car has traveled */
int vision[100][3];                /* 2-D array to hold vision file */
int eye              = 0;          /* loop counter and elements in vision */
int numsights;
int lastremembered   = 10;         /* the driver remembers his last assigned
                                      speed */
int numpoints;                     /* number of points in roadmap */
int stopcount, cyclecount;         /* control variables for arrays holding
                                      random numbers */
int brakeposition    = OFF;
int accel_brake      = 0;          /* acceleration due to braking */
int savedtime        = 0;
int brakedata[3000];               /* stores test data */
int PLANNED_DIST;
int cmdvel[1000];

Boolean stopping     = FALSE;

Boolean firstcall    = TRUE;
float velocity       = 0.0;        /* velocity of the car   */
float brakingdist;
float PLANNED_VELOCITY;
float cx = 0.0;                    /* car's x coordinate */
float cy = 0.0;                    /* car's y coordinate  */
float data[3000][2];               /* array to store data .*/
float est_i[3000];                 /* holds ksubi  */
float est_f[3000];                 /* holds ksubf  */
float ksubi[10], ksubf[10], ksube[150], ksubn[150];
```

```
FILE *fopen(), *fp;

Tag cartag, greenlighttag, yellowlighttag, redlighttag;
Tag cmdvelocitytag, carvelocitytag, brakepositiontag;
Tag laptag, modetag, unittag, tentag, hundredtag, thoutag;
Tag modeinserttag, timetag, testtag;

Object mapobj, gauges;

Device keypressed;

/*********************************************

filename: VARS.EXT.H
author: Michael J. Dolezal
date: May 20, 1987

/*********************************************
#include "gl.h"
#include "const.h"
#include "stdio.h"

extern int cmdvelocity;              /* commanded vehicle velocity */
extern int cruisevelocity;           /* read from vision file  */
extern int distance;                 /* distance the car has traveled */
extern int vision[100][3];           /* 2-D array to hold vision file */
extern int eye, numsights;           /* loop counter and elements in vision */
extern int lastremembered;           /* the driver remembers his last assigned
                                        speed  */
extern int numpoints;                /* number of points in roadmap  */
extern int stopcount, cyclecount;    /* control variables for the random
                                        number arrays   */
extern int brakeposition;
extern int accel_brake;
extern int savedtime;
extern int brakedata[3000];          /* stores test data  */
extern int PLANNED_DIST;

extern Boolean stopping;

extern Boolean firstcall;

extern float velocity;               /* velocity of the car   */
```

```
extern float brakingdist;
extern float PLANNED_VELOCITY;
extern float ksubi[10], ksubf[10], ksube[150], ksubn[150];
extern float data[3000][2];
extern float est_i[3000];
extern float est_f[3000];

extern FILE *fopen(), *fp;

extern Tag cartag, greenlighttag, yellowlighttag, redlighttag;
extern Tag cmdvelocitytag, carvelocitytag, brakepositiontag;
extern Tag laptag, modetag, unittag, tentag, hundredtag, thoutag;
extern Tag modeinserttag, timetag, testtag;

extern Device keypressed;

/*********************************************

filename: VISION.H
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

4 4 95
40 4 52
292 6 0
300 4 75
1135 5 0
1136 4 65
1239 4 50
1331 5 0
1340 4 55

/*********************************************

filename: MAKEFILE
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

FLAGS = -g
```

```
SRCS = navigator.c\
       clear.c\
       welcome.c\
       generate.c\
       cruise.c\
       mouse.c\
       checkkey.c\
       savedata.c\
       loadarray.c\
       mapview.c\
       gauges.c\
       brake.c\
       signal.c\
       netV.c

OBJS = navigator.o\
       clear.o\
       welcome.o\
       generate.o\
       cruise.o\
       mouse.o\
       checkkey.o\
       savedata.o\
       loadarray.o\
       mapview.o\
       gauges.o\
       signal.o\
       brake.o\
       netV.o

nav: (OBJS)cc -o nav (OBJS) -Zf -Zg -lm -g -lbsd -ldbm
(OBJS) : const.h .ps 12
```

APPENDIX B

SOURCE CODE FOR THE DRIVER'S DISPLAY

```
/*********************************************

filename: CARSIMU.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

This is the main program of the entire vehicle simulation
program(original by Tan Chiam Huat, modified by Mike Dolezal).
To recompile this program just issue the command "make".
This program allows a user on an IRIS workstation to send
data to and receive control information from another user
on the UNIX1 VAX.

*/

#include "const.h"
#include "vars.h"
#include "stdio.h"
#include "errno.h"
#include "sys/signal.h"

main()
{

/*************************************************************

        L O C A L       D E C L A R A T I O N S

*************************************************************/

int mousex, cal_mousex;
int cmdbrakeposition        = OFF; /* brake info networked from controller */
int remotedriver            = 0;   /* decoded steering signal */
```

```
int prev_mousex = 250;
int prevremotedriver        = 250; /* previous input from remote driver */
int sampling_interval       = 1;
int old_sampling_cycle      = -1;
int new_sampling_cycle;
int lightcolor = REDLIGHT;          /* used to pass signallight to driver  */
int cardriver;                     /* socket to the local client system  */
int nbyte;                          /* read/write variable */
int connect_server();              /* function connecting car to the driver */
int no_coord;
int where                    = 0;
int counter                  = 0;  /* counter for signal light timing  */
int lap    = 0;                     /* lap counter for arctan function  */
int i, count, unit, ten, hundred, thousand, no_of_round;
int mode                     = 0;  /* current operating mode */
int status                   = 0;
int condition                = 0;  /* current operating mode */
int brakeposition            = 0;
int accel_brake              = 0;  /* acceleration due to braking  */
int command                  = 0;  /* received from driver  */
                                   /* temp vars to minimize function calls  */
int statussize, cxsize, czsize, distancesize, commandsize,
    controlsize, velocitysize, carstatsize;

long carstat                 = 0;   /* used to send the velocity and the
                                       brakeposition to the recorder */
long controlsignal           = 0;   /* steering and brake info received from
                        controller  */
float cmdspeed               = 0.0;
float prediction_distance;
float tolerance              = 1.0;
float old_sigma              = 0.4;
                                   /* used to correct discontinous arctangent */
float lastsigma              = 0.0;
float temp, temp1, tempvel;
float gx, gy, gz;

char thousandc[2], hundredc[2], tenc[2], unitc[2];
char timec[10];                    /*  Car time in char format  */

extern long time();                /*  System clock  */
long clocktime;                    /*  For clock value */
char *clockc;
```

```
Boolean   alarm              = FALSE;      /* Off road warning flag */
Boolean   debug              = FALSE;      /* Turn off debug info */
Boolean   notdone            = TRUE;  /* Controls program termination */

Dimension consumption        = 1.0;     /* Fuel consumption */
Dimension crashdown          = 0.0;     /* Off-road display flag */
Dimension fueldown           = 0.0;     /* Fuel depleted display flag */
Dimension headingdeg         = 0.0;     /* Heading in degrees */
Dimension headingrad         = 0.0;     /* Heading in radians */
Dimension rdistance          = 0.0;     /* Distance travelled */
Dimension vd                 = 100.0;   /* Viewing distance   */
                                        /* Signal Light Setup */
Dimension timegreen   = 0.5;
Dimension timeyellow  = 1;
Dimension timered     = 4;

Coord crashx = 512.0;              /* X viewport coord to detect off-road */
Coord crashy = 385.0;              /* Y viewport coord to detect off-road */
Coord warnx1 = 212.0;              /* X viewport coord to warn off-road */
Coord warnx2 = 700.0;              /* X viewport coord to warn off-road */
Coord warny  = 385.0;                   /* Y viewport coord to warn off-road */

Colorindex colors[1];              /* Array to store color of crash spot */
short nopixel = 1;                 /* No of pixel to detect off-road */

Coord cx, cy, cz;                  /* Current viewing point */
Coord rx, ry, rz;                  /* Reference point   */
Coord pz, px;                      /* Last viewing point   */

Object terrain1, odometer, warning, heading_meter;
Object speedometer, fuel, steerwheel;
Object signboard, sky, mountain;
Object road, help, gauges, arrow, house, house1;

/*************************************************************

   S Y S T E M      I N I T I A L I Z A T I O N S

*************************************************************/

                                   /* Open up the net path to npscs-unix1  */
cardriver = connect_server("npscs-iris1", 5);
```

```
                                /* Initial state vector of the automobile */
state_vector[1] = 0.0;          /* initial z coord */
state_vector[2] = 0.0;          /* initial x coord */
state_vector[3] = 0.0;          /* initial velocity */
state_vector[4] = 0.0;          /* initial heading */
cx = 0.0;
cy = 3.0;
cz = 0.0;
rx = 0.0;
ry = 3.0;
rz = -vd;
                                /* complete function calls for read/writes  */
statussize = sizeof(status);
cxsize = sizeof(cx);
czsize = sizeof(cz);
distancesize = sizeof(rdistance);
commandsize = sizeof(command);
controlsize = sizeof(controlsignal);
velocitysize = sizeof(tempvel);
carstatsize = sizeof(carstat);
count = 0;
unit = ten = hundred = thousand = 0;
                                /* initialize signal lights */
timegreen = (timegreen * 20);
timeyellow = (timeyellow * 20) + timegreen;
timered = (timered * 20) + timeyellow;
                                /* initialize the workstation  */
ginit();
doublebuffer();
gconfig();
cursoff();
qdevice(KEYBD);
viewport(0, XMAXSCREEN, 0, YMAXSCREEN);
ortho2(0.0, 1023.0, 0.0, 767.0);
blink(10, CYAN, 255, 0, 0);
bbox2i(5, 5, 0, 1023, 0, 767);
                                /* Colors are further defined in const.h */
mapcolor(MOUNTAIN, 199, 123, 63);
mapcolor(MOUNTAIN1, 210, 150, 0);
mapcolor(FIELD, 5, 190, 20);
mapcolor(SKY, 50, 8, 155);
mapcolor(WARN, 125, 0, 0);
mapcolor(CHMWALL1,118,76,0);
```

```
mapcolor(CHMWALL2,146,114,0);
mapcolor(WINDOW,0,141,205);
mapcolor(SIDEROOF,188,50,14);
mapcolor(FRAME,118,50,14);
mapcolor(WALL,164,111,0);
mapcolor(SIDEWALL,146,94,1);
mapcolor(ROOF,148,50,14);
mapcolor(DIMGREEN,0, 110, 0);          /* colors for signal light */
mapcolor(DIMYELLOW, 170, 170, 0);
mapcolor(DIMRED, 110, 0, 0);
mapcolor(GRAY, 165, 165,165);
mapcolor(ROOF1,100,100,100);           /*  Dark Grey  */
mapcolor(FRAME1,0,60,60);              /*  Light Grey  */
mapcolor(SIDEWALL1,150,60,60);         /*  Light Grey  */
mapcolor(WALL1,160,60,60);             /*  Pink  */
setvaluator(MOUSEX, 250, 0, 500);      /* set the system mouse  */
setvaluator(MOUSEY, -10, -10, 400);
noise(MOUSEX, 10);

/*************************************************************

   M A K E    A L L    T H E    O B J E C T S

*************************************************************/
makethespeedometer(&speedometer);
makeheading(&heading_meter);
makesteerwheel(&steerwheel);
maketheodometer(&odometer);
maketerrain1(&terrain1);
makewarning(&warning);
maketheroad(&road);
makehouse1(&house1);
makehouse(&house);
makethesky(&sky);
makegauges(&gauges);
makehelp(&help);
makefuel(&fuel);

                                       /* Display the introductory image  */
    welcome();
                                       /* Read the roadmap  */
    no_coord = loadarray();
```

```
                                        /* Initialize the buffers  */
color(BLACK);
clear();
swapbuffers();
clear();
swapbuffers();

/*************************************************************

   M A I N     S I M U L A T I O N     L O O P

*************************************************************/

while(TRUE)
     {
     nbyte = read(cardriver, &command, commandsize);
     nbyte = read(cardriver, &controlsignal, controlsize);

     remotedriver = controlsignal/1000;
     condition = command/1000;
     cmdspeed = (command - (condition * 1000))/MPS_TO_KMPH;

                         /* adjust velocity for acceleration due to braking,
                            time is one cycle  */
     state_vector[3] = state_vector[3] + (BRAKEGAIN * accel_brake);
     if (state_vector[3] < 0.0) state_vector[3] = 0.0;

                         /*  counter for signal light color  */
     counter = counter + 1;
     new_sampling_cycle = count/sampling_interval;
     ++count;
     pz = cz; px = cx;
     clocktime = time((long *) 0);
     clockc = ctime(&clocktime);

                         /*  Sound alarm around 2m before off the road */
     cmov2(warnx1, warny);
     readpixels(nopixel, colors);
     if (colors[0] != BLACK && colors[0] != WHITE && colors[0] != YELLOW)
          alarm = TRUE;
     else alarm = FALSE;

     if (!alarm)
```

```
      {
      cmov2(warnx2, warny);
      readpixels(nopixel, colors);
      if (colors[0] != BLACK && colors[0] != WHITE &&
          colors[0] != YELLOW)
           alarm = TRUE;
      else alarm = FALSE;
      }

                  /* Check if the vehicle is off the road
                     IMPT : Assume road surface is black
                     and surface signs are white or yellow centerline   */
cmov2(crashx, crashy);
readpixels(nopixel, colors);
if (colors[0] != BLACK && colors[0] != WHITE && colors[0] != YELLOW)
      crashdown = -1000.0;

rz = - (vd*cos(state_vector[4]) + state_vector[1]);
rx = vd*sin(state_vector[4]) + state_vector[2];

                  /* prevent arctan from exploding by exceeding
                     small angle assumption when speed gets slow */
if (state_vector[3] < (10.0/MPS_TO_KMPH))
      {
      if (condition == ASteerDrSp) condition = DrManual;
      if (condition == ASteerNSp) condition = NavManual;
      if (condition == AUTOPILOT) condition = CruiseDrSteer;
      }

                  /* compute a new state for the vehicle statevectors. */
compute_new_state(condition);

cz = -state_vector[1];
cx = state_vector[2];

                  /*  Check if keyboard pressed.  */
checkkeybd(&notdone, &debug, &start, &mode, &condition);

                  /* combine data to improve efficiency */
status = (notdone * 100) + (lightcolor * 10) + mode;
tempvel = (state_vector[3] * MPS_TO_KMPH);

                  /* send data to the navigator  */
```

```
    nbyte = write(cardriver, &status, statussize);
    nbyte = write(cardriver, &tempvel, velocitysize);

    if (state_vector[3] > 0)
        {
        prediction_distance = state_vector[3] * prediction_time;
      /*
      "where" is passed to find_subgoal so that searching
      need not always start from the beginning of the road.
      The z and y convention in the graphics system is reversed.
      Also the sign is going in the opposite direction.  So
      compensate before passing into find_subgoal.
      */
    where = find_subgoal(no_coord, where, tolerance,
                         prediction_distance, cx, -cz, px, -pz, 0.0);
        }

switch(condition)
   {
   case ASteerDrSp: if (old_sampling_cycle < new_sampling_cycle)
                    {
                    old_sampling_cycle = new_sampling_cycle;
                    if (where < 0)
                       {
                           /* Stop completely and remove autopilot */
                       state_vector[3] = 0.0;
                       speed = 0.0;
                       condition = DrManual;
                       }
                     else
                        {
                        gx = roadmap[where][0];
                        gy = roadmap[where][1];
                        gz = roadmap[where][2];
                        }
                     }

       /* Convention difference: Z-axis in graphics is Y-axis in
       mathematical model.  Also Z-axis is negative when moving
       into the screen which therefore must be converted to
       positive for our calculation. */

       temp = -cz;
```

```
        sigma = atan2((gx-cx),(gy-temp));
        if (sigma - lastsigma < - 3.5)
              {
              lap = 1 + (int)((lastsigma - sigma - PI)/(2 * PI));
              sigma = sigma + 2 * PI * lap;
              lastsigma = sigma;
              }
              else
                 {
                 lastsigma = sigma;
                  }
        sigma_dot = (sigma - old_sigma)/deltat;
        old_sigma = sigma;
        if (getbutton(MOUSE1))
              {
              if (speed < (98.0/MPS_TO_KMPH))
                    {
                    start = FALSE;
                    speed = speed + (speedinc/MPS_TO_KMPH);
                    }
                    else  speed = 100.0/MPS_TO_KMPH;  /*  Top Speed */
              }
        if (getbutton(MOUSE2)) /* decrease speed */
              {
              speed = speed - speedinc;
              if (speed < 0.0) speed = 0.0;
              }
        if (getbutton(MOUSE3)) /* decrease speed */
              {
              speed = 0.0;
              }
        brakeposition = (getvaluator(MOUSEY))/2;
        if (brakeposition < 0) brakeposition = 0;
        accel_brake = -brakeposition;

        carstat = (brakeposition * 1000) + (int)(speed);

        nbyte = write(cardriver, &carstat, carstatsize);
        break;

case AUTOPILOT:
case ASteerNSp: if (old_sampling_cycle < new_sampling_cycle)
```

```
                    {
                    old_sampling_cycle = new_sampling_cycle;
                    if (where < 0)
                       {
                            /* Stop completely and remove autopilot */
                       state_vector[3] = 0.0;
                       speed = 0.0;
                       condition = DrManual;
                       }
                     else
                        {
                        gx = roadmap[where][0];
                        gy = roadmap[where][1];
                        gz = roadmap[where][2];
                        }
                    }
    /* Convention difference: Z-axis in graphics is Y-axis in
    mathematical model.  Also Z-axis is negative when moving
    into the screen which therefore must be converted to
    positive for our calculation. */

      temp = -cz;
      sigma = atan2((gx-cx),(gy-temp));
      if (sigma - lastsigma < - 3.5)
         {
         lap = 1 + (int)((lastsigma - sigma - PI)/(2 * PI));
         sigma = sigma + 2 * PI * lap;
         lastsigma = sigma;
         }
         else
            {
            lastsigma = sigma;
            }
      sigma_dot = (sigma - old_sigma)/deltat;
      old_sigma = sigma;
      speed =  cmdspeed;
      accel_brake = -(controlsignal - (remotedriver * 1000));
      break;
               /* Nav speed and Driver's steering */
case DrSteerNavSp:
               /* cruisecontrol and local steering */
case CruiseDrSteer: mousex = getvaluator(MOUSEX);
      cal_mousex = mousex - prev_mousex;
```

```
        steer_wheel_angle = steer_wheel_angle + (float) cal_mousex/200;
        prev_mousex = mousex:
        speed = cmdspeed;
        accel_brake = -(controlsignal - (remotedriver * 1000));
        start = FALSE;
        break;
                /* steering and speed with local mouse  */
case DrManual:  mousex = getvaluator(MOUSEX);
        cal_mousex = mousex - prev_mousex;
        steer_wheel_angle = steer_wheel_angle + (float) cal_mousex/200;
        prev_mousex = mousex;
        if (getbutton(MOUSE1))
             {
             if (speed < (98.0/MPS_TO_KMPH))
                  {
                  speed = speed + (speedinc/MPS_TO_KMPH);
                  }
                  else  speed = 100.0/MPS_TO_KMPH;  /*  Top Speed */
             }
        if (getbutton(MOUSE2)) /* decrease speed */
             {
              speed = speed - speedinc;
             if (speed < 0.0) speed = 0.0;
             }
        if (getbutton(MOUSE3)) /* decrease speed */
             {
             speed = 0.0;
             }
        brakeposition = (getvaluator(MOUSEY))/2;
        if (brakeposition < 0) brakeposition = 0;
        accel_brake = - brakeposition;

        carstat = (brakeposition * 1000) + (int)(speed);

        nbyte = write(cardriver, &carstat, carstatsize);

        start = FALSE;
        break;

                /* steering and speed with mouse on remote controller */
                /* cruise control and remote steering */
case CruiseNavSteer:
case NavManual: cal_mousex = remotedriver - prevremotedriver;
```

```
        steer_wheel_angle = steer_wheel_angle + (float) cal_mousex/200;
        prevremotedriver = remotedriver;
        speed = cmdspeed;
        accel_brake = -(controlsignal - (remotedriver * 1000));
        start = FALSE;
        break;

case NavSteerDrSp: cal_mousex = remotedriver - prevremotedriver;
        steer_wheel_angle = steer_wheel_angle + (float) cal_mousex/200;
        prevremotedriver = remotedriver;
        if (getbutton(MOUSE1))
              {
              if (speed < (98.0/MPS_TO_KMPH))
                    {
                    speed = speed + (speedinc/MPS_TO_KMPH);
                    }
                    else  speed = 100.0/MPS_TO_KMPH;  /*  Top Speed */
              }
        if (getbutton(MOUSE2)) /* decrease speed */
              {
              speed = speed - speedinc;
              if (speed < 0.0) speed = 0.0;
              }
        if (getbutton(MOUSE3)) /* decrease speed */
              {
              speed = 0.0;
              }
        brakeposition = (getvaluator(MOUSEY))/2;
        if (brakeposition < 0) brakeposition = 0;
        accel_brake = -brakeposition;
        carstat = (brakeposition * 1000) + (int)(speed);

        nbyte = write(cardriver, &carstat, carstatsize);

        start = FALSE;
        break;
} /* end switch  */

                    /*  Clear the vehicle window  */
  viewport(0, XMAXSCREEN, 385, YMAXSCREEN);
  color(FIELD);
  clear();
```

```
                    /*  Clear the display panel  */
viewport(0, XMAXSCREEN, 0, 380);
color(WHITE);
clear();

                    /*  Reset viewport  */
viewport(0, XMAXSCREEN, 0, YMAXSCREEN);

                    /*  Calculate distance travelled  */

rdistance = rdistance+sqrt((cz-pz)*(cz-pz)+(cx-px)*(cx-px));
distance = (int) rdistance;

nbyte = write(cardriver, &cx, cxsize);
nbyte = write(cardriver, &cz, czsize);
nbyte = write(cardriver, &rdistance, distancesize);

thousand = distance/1000;
hundred = (distance - thousand*1000)/100;
ten = (distance - hundred * 100 - thousand*1000)/10;
unit = distance - ten * 10  - hundred * 100 - thousand*1000;

if (unit == 10) { unit = 0; ++ten; }
if (ten == 10) { ten = 0; ++hundred; }
if (hundred == 10) { hundred = 0; ++thousand; }
if (thousand == 10) thousand = 0;
sprintf(timec,"%5.2f",car_time);
sprintf(thousandc,"%d",thousand);
sprintf(hundredc,"%d",hundred);
sprintf(tenc,"%d",ten);
sprintf(unitc,"%d",unit++);

                    /* DISPLAY HELP PANEL  */
callobj(help);

                    /*  EDIT SKY  */
editobj(sky);
objreplace(skylooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
closeobj();
callobj(sky);

                    /*  EDIT TERRAIN  */
```

```
editobj(terrain1);
objreplace(terrain1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
closeobj();
callobj(terrain1);

                    /*  EDIT ROAD  */
editobj(road);
objreplace(roadlooktag);
lookat(cx, cy, cz, rx, ry, rz, 0.0);
if (counter < timegreen)
    {
    objreplace(greenlighttag);
    color(GREEN);
    objreplace(redlighttag);
    color(DIMRED);
    lightcolor = GREENLIGHT;
    }
    else if ((counter > timegreen) && (counter <= timeyellow))
        {
        objreplace(greenlighttag);
        color(DIMGREEN);
        objreplace(yellowlighttag);
        color(YELLOW);
        lightcolor = YELLOWLIGHT;
        }
        else if ((counter > timeyellow) && (counter <= timered))
            {
            objreplace(yellowlighttag);
            color(DIMYELLOW);
            objreplace(redlighttag);
            color(RED);
            lightcolor = REDLIGHT;
            }
            else if (counter > timered)
                {
                counter = 0;
                }
  closeobj();
callobj(road);

                    /*  EDIT HOUSES  */
editobj(house);
```

```
objreplace(houselooktag);
lookat(cx,cy,cz,rx.ry.rz,0.0);
objreplace(housetranstag);
translate(-80.0, 0.0, -50.0);
objreplace(housescaletag);
scale(0.40, 0.40, 1.0);
closeobj();
callobj(house);

editobj(house1);
objreplace(house1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(house1transtag);
translate(-30.0, 0.0, -10.0);
objreplace(house1scaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house1);

editobj(house1);
objreplace(house1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(house1transtag);
translate(-30.0, 0.0, -15.0);
objreplace(house1scaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house1);
                    /*  EDIT STEERING WHEEL  */
editobj(steerwheel);
objreplace(steerwheeltag);
rotate((int) -(steer_wheel_angle * 10 * RAD_TO_DEG), 'Z');
closeobj();
callobj(steerwheel);
                    /*  EDIT ODOMETER  */
editobj(odometer);
objreplace(odotag1);
charstr(thousandc);
objreplace(odotag2);
charstr(hundredc);
objreplace(odotag3);
charstr(tenc);
objreplace(odotag4);
```

```
charstr(unitc);
closeobj();
callobj(odometer);

                    /* Display the clock  */
color(WHITE);
cmov2i(100, 750);
charstr(clockc);
color(BLACK);
color(CYAN);
cmov2i(400, 750);
  switch(condition)
     {
     case AUTOPILOT: charstr("AutoPilot");
                     break;
     case CruiseDrSteer:  charstr("Cruise Control, Driver's Steering");
                     break;
     case CruiseNavSteer:  charstr("Cruise Control, Navigator's Steering");
                     break;
     case ASteerDrSp:  charstr("AutoSteer, Driver's Speed");
                     break;
     case ASteerNSp:  charstr("AutoSteer, Navigator's Speed");
                     break;
     case DrManual:   charstr("Driver Manual Control");
                     break;
     case NavManual:   charstr("Navigator Manual Control");
                     break;
     case DrSteerNavSp: charstr("Driver Steers, Nav's Speed");
                     break;
     case NavSteerDrSp: charstr("Nav Steers, Driver's Speed");
                     break;
     }
color(BLACK);

                    /*  EDIT WARNING INDICATOR  */

  if (accel_brake != OFF)
        {
        editobj(warning);
        objreplace(braketag);
        color(RED);
        }
        else
```

```
        {
        editobj(warning);
        objreplace(braketag);
        color(WARN);
        }
if (state_vector[3] > 0)
      {
      objreplace(belttag);
      color(WARN);
      if (alarm)
            {
            objreplace(dangertag);
            color(CYAN);
            }
            else
                {
                objreplace(dangertag);
                color(WARN);
                }
      closeobj();
      }
      else
                  /*  BRAKE SIGNAL FOR CAR STOP  */
          {
          editobj(warning);
          objreplace(braketag);
          color(RED);
          objreplace(temptag);
          color(RED);
          closeobj();
          }
if ((count < 1000) && (state_vector[3] > 0.0))
      {
      editobj(warning);
      objreplace(temptag);
      color(WARN);
      closeobj();
      }
if ((count > 5000) && (state_vector[3] * MPS_TO_KMPH > 65))
      {
      editobj(warning);
      objreplace(temptag);
      color(RED);
```

```
        closeobj();
        }
  callobj(warning);
                        /*  EDIT HEADING INDICATOR  */
                        /*  Compute heading using vehicle state vector  */
  if (state_vector[4] < 0.0) headingrad = 2*PI+state_vector[4];
        else headingrad = state_vector[4];
  no_of_round = (headingrad*180.0/PI)/360.0;
  headingdeg = headingrad*180.0/PI - (float) no_of_round*360;

  editobj(heading_meter);
  objreplace(transl1);
  translate(heading_xpos-20.0-4.5*headingdeg, 4.0, 0.0);
  closeobj();
callobj(heading_meter);
                        /*  EDIT SPEEDOMETER INDICATOR  */
                    /*  2.5 factor is for converting to the dashboard display */

speedbar = 181.0 - state_vector[3] * MPS_TO_KMPH * 2.5;
  editobj(speedometer);
objreplace(transl4);
translate(0.0, speedbar, 0.0);
closeobj();
callobj(speedometer);
                        /* EDIT BRAKE AND CMDSPEED GUAGES */
editobj(gauges);
objreplace(manbraketag);
rectfi(BRAKEX, BRAKEY, BRAKEX + 50, BRAKEY - accel_brake);
objreplace(manspeedtag);
rectfi(CMDX, CMDY, CMDX + 50,
   CMDY + (int)(2 * speed * MPS_TO_KMPH));
closeobj();
callobj(gauges);
                        /* EDIT FUEL GUAGE  */
if (state_vector[3] * MPS_TO_KMPH > 0.0)
    {
    fuelquant = fuelquant - consumption;
    }
    if (fuelquant <= 0.0)
        {
        fueldown = -1000.0;
        }
  fuelbar = fuelquant/MAXFUEL*320.0 + 14.0;
```

```
    editobj(fuel);
    objreplace(fuel1);
    rectf(281.0, 14.0, 324.0, fuelbar);
    closeobj();
    callobj(fuel);
                    /* EDIT CRASH INFO DISPLAY FOR OFF-ROAD */
     pushmatrix();
     pushattributes();
     translate(0.0,crashdown,0.0);

                          /*  Set all warning lights when crash  */
    if (crashdown == -1000)
        {
        editobj(warning);
        objreplace(braketag);
        color(RED);
        objreplace(dangertag);
        color(RED);
        closeobj();
        callobj(warning);
        }
     color(RED);
     rectf(0.0,1385.0,1023.0,1767.0);

    color(BLACK);
    cmov2i(370,1576);
    charstr("CRASH");
     cmov2i(370,1560);
    charstr("OFF THE ROAD");

     cmov2i(370,1544);
    charstr("PUSH 'E' TO EXIT");

     popattributes();
     popmatrix();

    swapbuffers();
    } /* while loop */

close(cardriver);
color(BLACK);
clear();
swapbuffers();
```

```
clear();
swapbuffers();
finish();
gexit();
} /* main */

/*********************************************

filename: CIRCUIT.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"

/*************************************************************

    B U I L D    T H E    R A L L Y    C I R C U I T

*************************************************************/

maketheroad(road)
Object *road;
{
Dimension  temp, i;
Dimension  high       = 3.2;
Colorindex signbg     = YELLOW;
Colorindex upsign     = RED;
Colorindex rightsign = BLUE;
*road = genobj();
makeobj(*road);
pushmatrix();
pushviewport();
viewport(0, XMAXSCREEN, 385, YMAXSCREEN);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
roadlooktag = gentag();
maketag(roadlooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);
```

```
/*****************************

    FIRST STRETCH OF ROAD

*****************************/

surf(0.0, 0.0, 0.0, ROADWIDTH, ROADLEN, BLACK);
surf(-4.2, 0.0, 0.0, 0.2, ROADLEN, YELLOW);

/*

Build the sign "START"

*/

temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp + 1.0, 0.0, 0.0);
rotate(-900,'X');
letter('T', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 0.25, 0.0, 0.0);
rotate(-900,'X');
letter('R', BLACK);
popmatrix();
color(WHITE);
pushmatrix();
translate(temp - 1.5, 0.0, 0.0);
rotate(-900,'X');
letter('A', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, 0.0);
rotate(-900,'X');
letter('T', BLACK);
popmatrix();
```

```
color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, 0.0);
rotate(-900,'X');
letter('S', BLACK);
popmatrix();

/*

Build the sign "TURN" before the bend

*/

color(WHITE);
pushmatrix();
translate(-3.25, 0.0, -(ROADLEN - 5.0));
rotate(-900,'X');
letter('N', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(-4.5, 0.0, -(ROADLEN - 5.0));
rotate(-900,'X');
letter('R', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(-5.75, 0.0, -(ROADLEN - 5.0));
rotate(-900,'X');
letter('U', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(-7.0, 0.0, -(ROADLEN - 5.0));
rotate(-900,'X');
letter('T', BLACK);
popmatrix();


/*
```

```
Add the first crossroad

*/

pushmatrix();
translate(0.0, 0.0, -(ROADLEN/4.0));
rotate(900, 'Y');
translate(0.0, 0.0, 2.0 * ROADLEN);
stripe(0.0, 0.0, 0.0, ROADWIDTH/2.0, 8.0 * ROADLEN, BLACK);
for (temp = 20.0; temp < (4.0 * ROADLEN); temp += 40.0)
    {
    pushmatrix();
    translate(0.0, 0.0, -temp);
    rotate(-900,'X');
    polyarrow(0.7, 1.2, 0.0, WHITE);
    popmatrix();
    }
popmatrix();

/*

Add the second crossroad

*/

pushmatrix();
translate(0.0, 0.0, -(3.0 * ROADLEN/4.0));
rotate(900, 'Y');
translate(0.0, 0.0, 2.0 * ROADLEN);
stripe(0.0, 0.0, 0.0, ROADWIDTH/2.0, 8.0 * ROADLEN, BLACK);
for (temp = 20.0; temp < (4.0 * ROADLEN); temp += 40.0)
    {
    pushmatrix();
    translate(0.0, 0.0, -temp);
    rotate(-900,'X');
    polyarrow(0.7, 1.2, 0.0, WHITE);
    popmatrix();
    }
popmatrix();

/*

Create 1st speedlimit sign
```

```
*/

pushmatrix();
translate(6.0, 0.0, -15.0);
speedlimit('2', '5');
popmatrix();

/*

Create 2nd speedlimit sign

*/

pushmatrix();
translate(6.0, 0.0, -(ROADLEN/4.0) - 15);
speedlimit('3', '5');
popmatrix();

/*

Create the 50 meter distance marker

*/
pushmatrix();
translate(0.0, 0.0, -42.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, -38.25);
rotate(-900, 'X');
letter('O', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, -38.25);
rotate(-900, 'X');
letter('5', BLACK);
popmatrix();
```

```
/*

Create the 40 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -52.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, -48.25);
rotate(-900, 'X');
letter('O', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, -48.25);
rotate(-900, 'X');
letter('4', BLACK);
popmatrix();

/*

Create the 30 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -62.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();
temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, -58.25);
rotate(-900, 'X');
```

```
letter('O', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, -58.25);
rotate(-900, 'X');
letter('3', BLACK);
popmatrix();

/*

Create the 20 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -72.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, -68.25);
rotate(-900, 'X');
letter('O', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, -68.25);
rotate(-900, 'X');
letter('2', BLACK);
popmatrix();

/*

Create the 10 meter distance marker

*/
```

```
pushmatrix();
translate(0.0, 0.0, -82.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

temp = -3.6;
color(WHITE);
pushmatrix();
translate(temp - 2.75, 0.0, -78.25);
rotate(-900, 'X');
letter('O', BLACK);
popmatrix();
color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, -78.25);
rotate(-900, 'X');
letter('1', BLACK);
popmatrix();


/*

Create 1st stopsign

*/

pushmatrix();
translate(6.0, 0.0, -ROADLEN/4.0 + 5.0);
stopsign(1.9, 3.0);
popmatrix();

/*

Create 3rd speedlimit sign

*/

pushmatrix();
translate(6.0, 0.0, -(3.0 * ROADLEN/4.0) - 10);
speedlimit('5', 'O');
popmatrix();
```

```
/*

Create the 60 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -232.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

pushmatrix();
translate(5.2, 0.0, -232.0);
billboard('6', 'O');
popmatrix();

/*

Create the 50 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -242.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

pushmatrix();
translate(5.2, 0.0, -242.0);
billboard('5', 'O');
popmatrix();
/*

Create the 40 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -252.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
```

```
popmatrix();

pushmatrix();
translate(5.2, 0.0, -252.0);
billboard('4', 'O');
popmatrix();

/*

Create the 30 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -262.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

pushmatrix();
translate(5.2, 0.0, -262.0);
billboard('3', 'O');
popmatrix();

/*

Create the 20 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -272.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();

pushmatrix();
translate(5.2, 0.0, -272.0);
billboard('2', 'O');
popmatrix();

/*
```

```
Create the 10 meter distance marker

*/

pushmatrix();
translate(0.0, 0.0, -282.0);
rotate(-900, 'Y');
stripe(0.0, 0.0, 0.0, 0.2, 4.0, WHITE);
popmatrix();
pushmatrix();
translate(5.2, 0.0, -282.0);
billboard('1', 'O');
popmatrix();

/*

Create the signals

*/

pushmatrix();
translate (6.0, 0.0, -((3.0 * ROADLEN)/4.0) - 5.0);
signallight(12.0);
popmatrix();


/*

Create 2nd uparrow signboard

*/

pushmatrix();
translate(-12.0, 0.0, -(ROADLEN/3.0));
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(-12.0, high, -(ROADLEN/3.0));
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*
```

```
First road bend

*/

pushmatrix();
translate(BENDRADIUS, 0.0, -ROADLEN);
rotate(-900, 'X');
bend();
popmatrix();

/*

Build 1st right turn signboard

*/

pushmatrix();
translate(5.0, 0.0, -(ROADLEN-5.0));
signb(1.9, 2.5, 3.0. signbg);
popmatrix();
pushmatrix();
translate(4.3, 4.0, -(ROADLEN-5.0));
rotate(-900,'Z');
polyarrow(0.7, 1.2, 0.0, rightsign);
popmatrix();
/*******************************

    SECOND STRETCH OF ROAD

*******************************/

pushmatrix();
translate(BENDRADIUS, 0.0, -ROADLEN - BENDRADIUS);
rotate(-900, 'Y');
surf(0.0, 0.0, 0.0, ROADWIDTH, ROADLEN, BLACK);
popmatrix();

/*

Build a series of road strips

*/
```

```
color(WHITE);
for (i = BENDRADIUS + 8.0; i < ROADLEN; i += 20.0)
     {
     pushmatrix();
     translate(i, 0.0, -ROADLEN - BENDRADIUS - (ROADWIDTH/4));
     rotate(-900, 'Z');
     rotate(-900, 'Y');
     rectf(-0.5, 0.0, 0.5, 3.0);
     popmatrix();
     }
color(BLACK);

/*

Create 3rd uparrow signboard

*/

pushmatrix();
translate(BENDRADIUS + 50.0, 0.0, -ROADLEN - BENDRADIUS + 6);
rotate(-900,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(BENDRADIUS + 50.0, high, -ROADLEN - BENDRADIUS + 6);
rotate(-900,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*

Second road bend

*/

pushmatrix();
temp = BENDRADIUS + ROADLEN;
translate(temp, 0.0, -ROADLEN);
rotate(-900, 'X');
rotate(-900, 'Z');
bend();
popmatrix();
/*
```

```
Build 2nd right turn signboard

*/

pushmatrix();
translate(temp - ROADWIDTH, 0.0,
        -ROADLEN - BENDRADIUS - (3 * ROADWIDTH/4.0) - 2);
rotate(-900,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(temp - ROADWIDTH, 4.0,
        -ROADLEN - BENDRADIUS - 2.7 - (3 * ROADWIDTH/4.0));
rotate(-900,'Y');
rotate(-900,'Z');
polyarrow(0.7, 1.2, 0.0, rightsign);
popmatrix();


/*******************************

    THIRD STRETCH OF ROAD

*******************************/


pushmatrix();
temp = 2 * BENDRADIUS + ROADLEN;
translate(temp + (ROADWIDTH/2), 0.0, 0.0);
surf(0.0, 0.0, 0.0, ROADWIDTH, ROADLEN, BLACK);
popmatrix();

/*

Create a series of arrows

*/

for (i = ROADLEN; i > 5.0; i -= 20.0)
      {
      pushmatrix();
      translate(temp + (ROADWIDTH/4.0), 0.0, -i);
      rotate(-900,'X');
```

```
        rotate(-1800, 'Z');
        polyarrow(0.7, 1.2, 0.0, WHITE);
        popmatrix();
        }

/*

Create 5th speedlimit sign

*/

pushmatrix();
translate(ROADLEN + (2 * BENDRADIUS) - 6.0,
        0.0, -ROADLEN/4.0 + 10.0);
rotate(-1800, 'Y');
speedlimit('5','5');
popmatrix();

/*  Create 3rd stopsign */

pushmatrix();
translate(ROADLEN + (2 * BENDRADIUS) - 6.0, 0.0, -ROADLEN/4.0 - 5.0);
rotate(-1800, 'Y');
stopsign(1.9, 3.0);
popmatrix();

/*

Create 4nd uparrow signboard

*/

pushmatrix();
translate(temp + ROADWIDTH, 0.0, -ROADLEN + 10.0);
rotate(-1800,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(temp + ROADWIDTH, high, -ROADLEN + 10.0);
rotate(-1800,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();
```

```
/*

Create 5th speedlimit sign

*/

pushmatrix();
translate(ROADLEN + (2 * BENDRADIUS) - 6.0,
    0.0, - 3 * ROADLEN/4.0 + 10.0);
rotate(-1800, 'Y');
speedlimit('4','5');
popmatrix();

/*

Create 2nd stopsign

*/

pushmatrix();
translate(ROADLEN + (2 * BENDRADIUS) - 6.0,
        0.0, - 3 * ROADLEN/4.0 - 5.0);
rotate(-1800, 'Y');
stopsign(1.9, 3.0);
popmatrix();

/*

Third road bend

*/

pushmatrix();
temp = BENDRADIUS + ROADLEN;
translate(temp, 0.0, 0.0);
rotate(-900, 'X');
rotate(-1800, 'Z');
bend();
popmatrix();
/*****************************

    FOURTH STRETCH OF ROAD
```

```
*******************************/

pushmatrix();
temp = BENDRADIUS;
translate(temp, 0.0, temp + (ROADWIDTH/2));
rotate(-900, 'Y');
surf(0.0, 0.0, 0.0, ROADWIDTH, ROADLEN, BLACK);
popmatrix();

/*

Create a series of arrows

*/

for (i = temp + 10.0; i < temp + ROADLEN; i += 20.0)
      {
      pushmatrix();
      translate(i, 0.0, temp + (ROADWIDTH/4.0));
      rotate(-900,'X');
      rotate(900,'Z');
      polyarrow(0.7, 1.2, 0.0, WHITE);
      popmatrix();
      }

/*

Create 5nd uparrow signboard

*/

pushmatrix();
translate(ROADLEN, 0.0, BENDRADIUS + ROADWIDTH);
rotate(-2700,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(ROADLEN, high, BENDRADIUS + ROADWIDTH);
rotate(-2700,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*
```

```
Fourth road bend

*/

pushmatrix();
translate(BENDRADIUS, 0.0, 0.0);
rotate(-900, 'X');
rotate(900, 'Z');
bend();
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* maketheroad */

/*********************************************

filename: OTHER.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"


/*************************************************************

                        S K Y

*************************************************************/

makethesky(sky)
Object *sky;
{
*sky= genobj();
makeobj(*sky);

pushmatrix();
pushviewport();
```

```
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);

skylooktag = gentag();
maketag(skylooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
translate(0.0, 100.0, 50000.0);
surf(0.0, 0.0, 0.0, 400000.0, 100000.0, SKY);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makethesky */

/*************************************************************

            C L O U D S   A N D   M O U N T A I N S

*************************************************************/

maketerrain1(terrain1)
Object *terrain1;
{
Dimension temp = -(ROADLEN+500.0);
Dimension temp1 = -(ROADLEN+500.0);
Dimension tempy = 0.0;
Dimension tempy1 = 100.0;
Dimension tempy2 = 350.0;

*terrain1= genobj();
makeobj(*terrain1);
/*  Generate some clouds  */

pushmatrix();
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
terrain1looktag= gentag();
```

```
maketag(terrain1looktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
color(WHITE);
translate(-1000.0, tempy1, temp);
scale(1.0, 1.0, 1.0);
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-1000.0, tempy1, temp);
scale(1.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-2000.0, tempy1, temp);
scale(2.0, 2.0, 1.0);
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-2000.0, tempy1, temp);
scale(2.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();
pushmatrix();
color(WHITE);
translate(2000.0, tempy1, temp);
```

```
scale(3.0, 2.0, 1.0);
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(2000.0, tempy1, temp);
scale(2.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

/* Generate some mountains */

pushmatrix();
translate(-2000.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 400.0, 0, 1800);
popmatrix();

pushmatrix();
translate(-1500.0, tempy, temp);
scale (1.0, 0.2, 0.0);
color(MOUNTAIN);
arcf(0.0, 0.0, 250.0, 0, 1800);
popmatrix();

pushmatrix();
translate(-1000.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();

pushmatrix();
translate(1000.0, tempy, temp);
scale (1.0, 0.2, 0.0);
color(MOUNTAIN);
```

```
arcf(0.0. 0.0, 250.0, 0, 1800);
popmatrix();

pushmatrix();
translate(1500.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();

pushmatrix();
translate(2000.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();
popviewport();
popmatrix();
closeobj();
}  /* maketerrain1*/

/*************************************************************

                    B U I L D    S U R F A C E S

*************************************************************/

surf(x, y, z, width, length, roadcolor)
Coord x, y, z;
Dimension width, length;
Colorindex roadcolor;
{
Coord vertice[5][3];
Dimension temp;
temp = width/4;
vertice[0][0] = x;
vertice[0][1] = y;
vertice[0][2] = z;

vertice[1][0] = x - (3 * temp);
vertice[1][1] = y;
vertice[1][2] = z;
```

```
vertice[2][0] = x - (3 * temp);
vertice[2][1] = y;
vertice[2][2] = -length;

vertice[3][0] = x + temp;
vertice[3][1] = y;
vertice[3][2] = -length;

vertice[4][0] = x + temp;
vertice[4][1] = y;
vertice[4][2] = z;
color(roadcolor);
polf(5,vertice);
}    /* surf */

stripe(x, y, z, width, length, roadcolor)
Coord x, y, z;
Dimension width, length;
Colorindex roadcolor;
{
Coord vertice[5][3];
Dimension temp;
temp = width/2;

vertice[0][0] = x;
vertice[0][1] = y;
vertice[0][2] = z;

vertice[1][0] = x - temp;
vertice[1][1] = y;
vertice[1][2] = z;

vertice[2][0] = x - temp;
vertice[2][1] = y;
vertice[2][2] = -length;
vertice[3][0] = x + temp;
vertice[3][1] = y;
vertice[3][2] = -length;

vertice[4][0] = x + temp;
vertice[4][1] = y;
vertice[4][2] = z;
```

```
color(roadcolor);
polf(5,vertice);
}   /* stripe */

/*************************************************************

                    B U I L D     R O A D     B E N D S

*************************************************************/

bend()
{
color(BLACK);
arcfi(0, 0, (int) (BENDRADIUS + (3 * ROADWIDTH/4)), 900, 1800);
color(FIELD);
arcfi(0, 0, (int) (BENDRADIUS - (ROADWIDTH/4)), 900, 1800);
}

/*************************************************************

                    B U I L D     S I G N B O A R D

*************************************************************/

signb(width, length, height, bcolor)
Dimension width, length, height;
Colorindex bcolor;
{
Coord vertice[5][3];
Coord vertice1[5][3];
Dimension legwidth, temp1, temp2, temp3;
legwidth = 0.2;    /* size of supporting leg */
temp1 = length/2;
temp2 = length/4;
temp3 = legwidth/2;

vertice[0][0] = 0.0;
vertice[0][1] = height;
vertice[0][2] = 0.0;

vertice[1][0] = -temp1;
vertice[1][1] = height;
vertice[1][2] = 0.0;
```

```
vertice[2][0] = -temp1;
vertice[2][1] = width + height;
vertice[2][2] = 0.0;

vertice[3][0] = temp1;
vertice[3][1] = width + height;
vertice[3][2] = 0.0;

vertice[4][0] = temp1;
vertice[4][1] = height;
vertice[4][2] = 0.0;
color(bcolor);
polf(5,vertice);

                    /* Generate the supporting leg */
vertice1[0][0] = 0.0;
vertice1[0][1] = 0.0;
vertice1[0][2] = 0.0;

vertice1[1][0] = -temp3;
vertice1[1][1] = 0.0;
vertice1[1][2] = 0.0;

vertice1[2][0] = -temp3;
vertice1[2][1] = height;
vertice1[2][2] = 0.0;

vertice1[3][0] = temp3;
vertice1[3][1] = height;
vertice1[3][2] = 0.0;

vertice1[4][0] = temp3;
vertice1[4][1] = 0.0;
vertice1[4][2] = 0.0;

color(BLACK);
polf(5,vertice1);
}    /* signboard */

/*************************************************************
```

B U I L D S T O P S I G N

```
**********************************************************/

stopsign (width, height)
Dimension width, height;
{
Coord vertice [10][3];
Coord verticel [5][3];
Dimension legwidth, temp1, temp2, temp3, temp4;
legwidth = 0.2;          /* size of the supporting leg */
temp1 = 5 * width/24;
temp2 = width/2;
temp3 = legwidth/2;
temp4 = 7 * width/24;

                    /*  Make the sign */
vertice [0][0] = 0.0;
vertice [0][1] = height;
vertice [0][2] = 0.0;

vertice [1][0] = temp1;
vertice [1][1] = height;
vertice [1][2] = 0.0;

vertice [2][0] = temp2;
vertice [2][1] = temp4 + height;
vertice [2][2] = 0.0;

vertice [3][0] = temp2;
vertice [3][1] = temp4 + (2 * temp1) + height;
vertice [3][2] = 0.0;
vertice [4][0] = temp1;
vertice [4][1] = height + width;
vertice [4][2] = 0.0;

vertice [5][0] = - temp1;
vertice [5][1] = height + width;
vertice [5][2] = 0.0;

vertice [6][0] = - temp2;
vertice [6][1] = temp4 + (2 * temp1) + height;
vertice [6][2] = 0.0;

vertice [7][0] = - temp2;
```

```
vertice [7][1] = temp4 + height;
vertice [7][2] = 0.0;

vertice [8][0] = - temp1;
vertice [8][1] = height;
vertice [8][2] = 0.0;

vertice [9][0] = 0.0;
vertice [9][1] = height;
vertice [9][2] = 0.0;

color(RED);
polf (10, vertice);

                    /* Put the face on the stopsign  */

color(WHITE);
linewidth(2);
poly (10, vertice);

                    /* Put the letters STOP on the sign  */

color(WHITE);
pushmatrix();
translate(-0.6, 4.0, 0.0);
scale(0.5, 0.5, 1.0);
translate(-5.0, -3.8, 0.0);
letter('S', RED);
popmatrix();

color(WHITE);
pushmatrix();
translate(-0.2,  4.0, 0.0);
scale(0.5, 0.5, 1.0);
translate(-5.0, -3.8, 0.0);
letter('T', RED);
popmatrix();

color(WHITE);
pushmatrix();
translate(0.2, 4.0, 0.0);
scale(0.5, 0.5, 1.0);
translate(-5.0, -3.8, 0.0);
```

```
letter('O', RED);
popmatrix();

color(WHITE);
pushmatrix();
translate(0.6, 4.0, 0.0);
scale(0.5, 0.5, 1.0);
translate(-5.0, -3.8, 0.0);
letter('P', RED);
popmatrix();

                    /*    Build the supporting leg   */

color(GRAY);
rectf(-temp3, 0.0, temp3, height);
}  /* Stopsign */


/*****************************************************************

                        B U I L D   S I G N A L

*****************************************************************/

signallight(height)
Dimension height;
{
Dimension legwidth = 0.5;
Dimension separation = 20;
Dimension temp, temp1, temp2;
temp = 0.70 * height;
temp1 = 0.30 * height;
temp2 = legwidth/2;

                        /* Build the post */

color(GRAY);
rectf(-temp2, 0.0, temp2, temp - (0.05 * height));
rectf(-temp2 - separation, 0.0, temp2 - separation, temp - (0.05 * height));

                        /* Build the background and lights   */

color(BLACK);
```

```
rectf(-(temp1/3.0), temp - (0.05 * height), temp1/3.0, (1.05 * height));
rectf(-(temp1/3.0) - separation, temp - (0.05 * height),
        temp1/3.0 - separation, (1.05 * height));
greenlighttag = gentag();
maketag(greenlighttag);
color(GREEN);
circf(0.0, ((temp1/6.0) + temp), temp1/7.0);
circf(-separation, ((temp1/6.0) + temp), temp1/7.0);

yellowlighttag = gentag();
maketag(yellowlighttag);
color(DIMYELLOW);
circf(0.0, ((temp1/2.0) + temp), temp1/7.0);
circf(- separation, ((temp1/2.0) + temp), temp1/7.0);

redlighttag = gentag();
maketag(redlighttag);
color(DIMRED);
circf(0.0, height - (temp1/6.0), temp1/7.0);

circf(- separation, height - (temp1/6.0), temp1/7.0);
} /* Green Light */

/*************************************************************

                    B U I L D     A R R O W

*************************************************************/

polyarrow(bodywidth, headwidth, high, arrowcolor)
Colorindex arrowcolor;
Dimension bodywidth, headwidth, high;
{
Coord vertice[5][3], vertice1[3][3];
Dimension bodyheight = 0.8;
Dimension headheight = 1.5;
Dimension temp1 = bodywidth/2;
Dimension temp2 = headwidth/2;

vertice[0][0] = 0.0;
vertice[0][1] = 0.0 + high;
vertice[0][2] = 0.0;
```

```
vertice[1][0] = -temp1;
vertice[1][1] = 0.0 + high;
vertice[1][2] = 0.0;

vertice[2][0] = -temp1;
vertice[2][1] = bodyheight + high;
vertice[2][2] = 0.0;

vertice[3][0] = temp1;
vertice[3][1] = bodyheight + high;
vertice[3][2] = 0.0;

vertice[4][0] = temp1;
vertice[4][1] = 0.0 + high;
vertice[4][2] = 0.0;

color(arrowcolor);
polf(5,vertice);

vertice1[0][0] = -temp2;
vertice1[0][1] = bodyheight + high;
vertice1[0][2] = 0.0;

vertice1[1][0] = 0.0;
vertice1[1][1] = headheight + high;
vertice1[1][2] = 0.0;

vertice1[2][0] = temp2;
vertice1[2][1] = bodyheight + high;
vertice1[2][2] = 0.0;

color(arrowcolor);
polf(3,vertice1);
} /* polyarrow */

/*****************************************************************

                    B U I L D      H O U S E

*****************************************************************/

makehouse(house)
Object *house;
```

```
{
float sidewall[5][2], roof[4][2], chmwall1[4][2];
float chmwall2[4][2], sideroof[4][2];

*house=genobj();
makeobj(*house);

pushmatrix();
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
houselooktag = gentag();
maketag(houselooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
housetranstag = gentag();
maketag(housetranstag);
translate(0.0, 0.0, 0.0);
housescaletag = gentag();
maketag(housescaletag);
scale(1.0, 1.0, 1.0);

                        /* Draw front wall */

color(WALL);
rectf(-1.0,0.0,16.0,10.0);

                        /* Draw side wall */

sidewall[0][0]=(-4.0);
sidewall[0][1]=(2.0);
sidewall[1][0]=(0.0);
sidewall[1][1]=(0.0);
sidewall[2][0]=(0.0);
sidewall[2][1]=(10.0);
sidewall[3][0]=(-3.0);
sidewall[3][1]=(13.0);
sidewall[4][0]=(-4.0);
sidewall[4][1]=(11.5);

color(SIDEWALL);
```

```
polf2(5,sidewall);

                        /* Draw roof and sideroof */

roof[0][0]=(-1.0);
roof[0][1]=(10.0);
roof[1][0]=(17.0);
roof[1][1]=(10.0);
roof[2][0]=(14.0);
roof[2][1]=(13.5);
roof[3][0]=(-3.0);
roof[3][1]=(13.5);

color(ROOF);
polf2(4,roof);

sideroof[0][0]=(-4.3);
sideroof[0][1]=(11.5);
sideroof[1][0]=(-4.0);
sideroof[1][1]=(11.5);
sideroof[2][0]=(-2.8);
sideroof[2][1]=(13.1);
sideroof[3][0]=(-3.0);
sideroof[3][1]=(13.5);

color(SIDEROOF);
polf2(4,sideroof);

                        /* Draw window */

color(WINDOW);
rectf(2.0,4.0,5.0,7.0);
rectf(9.0,4.0,12.0,7.0);

                        /* Draw window frames */

color(FRAME);
linewidth(4);
move(2.0,4.0,0.0);
draw(5.0,4.0,0.0);
draw(5.0,7.0,0.0);
draw(2.0,7.0,0.0);
draw(2.0,4.0,0.0);
```

```
move(3.5,4.0,0.0);
draw(3.5,7.0,0.0);
move(2.0,5.5,0.0);
draw(5.0,5.5,0.0);

move(9.0,4.0,0.0);
draw(12.0,4.0,0.0);
draw(12.0,7.0,0.0);
draw(9.0,7.0,0.0);
draw(9.0,4.0,0.0);
move(10.5,4.0,0.0);
draw(10.5,7.0,0.0);
move(9.0,5.5,0.0);
draw(12.0,5.5,0.0);

                        /* Draw chimney front wall */

color(SIDEWALL);
rectf(1.0,12.0,3.0,14.2);

                        /* Draw the hole on the chimney */

color(BLACK);
rectf(1.5,13.3,2.5,13.8);

                        /* Draw top and side walls of the chimney */

chmwall1[0][0]=0.5;
chmwall1[0][1]=12.5;
chmwall1[1][0]=1.0;
chmwall1[1][1]=12.0;
chmwall1[2][0]=1.0;
chmwall1[2][1]=14.2;
chmwall1[3][0]=0.5;
chmwall1[3][1]=14.7;

color(CHMWALL1);
polf2(4,chmwall1);

chmwall2[0][0]=2.5;
chmwall2[0][1]=14.7;
chmwall2[1][0]=3.0;
chmwall2[1][1]=14.2;
```

```
chmwall2[2][0]=1.0;
chmwall2[2][1]=14.2;
chmwall2[3][0]=0.5;
chmwall2[3][1]=14.7;

color(CHMWALL2);
polf2(4,chmwall2);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makehouse */

makehouse1(house1)
Object *house1;
{
float sidewall[5][2], roof[4][2], chmwall1[4][2];
float chmwall2[4][2], sideroof[4][2];

*house1=genobj();
makeobj(*house1);

pushmatrix();
pushviewport();
viewport(0 - 260, 1023 - 260, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
house1looktag = gentag();
maketag(house1looktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
house1transtag = gentag();
maketag(house1transtag);
translate(0.0, 0.0, 0.0);
house1scaletag = gentag();
maketag(house1scaletag);
scale(1.0, 1.0, 1.0);

                    /* Draw front wall */
color(WALL1);
rectf(-1.0,0.0,16.0,10.0);
```

```
                        /* Draw side wall */
sidewall[0][0]=(-4.0);
sidewall[0][1]=(2.0);
sidewall[1][0]=(0.0);
sidewall[1][1]=(0.0);
sidewall[2][0]=(0.0);
sidewall[2][1]=(10.0);
sidewall[3][0]=(-3.0);
sidewall[3][1]=(13.0);
sidewall[4][0]=(-4.0);
sidewall[4][1]=(11.5);
color(SIDEWALL1);
polf2(5,sidewall);

                        /* Draw roof and sideroof */

roof[0][0]=(-1.0);
roof[0][1]=(10.0);
roof[1][0]=(17.0);
roof[1][1]=(10.0);
roof[2][0]=(14.0);
roof[2][1]=(13.5);
roof[3][0]=(-3.0);
roof[3][1]=(13.5);

color(ROOF1);
polf2(4,roof);

sideroof[0][0]=(-4.3);
sideroof[0][1]=(11.5);
sideroof[1][0]=(-4.0);
sideroof[1][1]=(11.5);
sideroof[2][0]=(-2.8);
sideroof[2][1]=(13.1);
sideroof[3][0]=(-3.0);
sideroof[3][1]=(13.5);

color(SIDEROOF);
polf2(4,sideroof);

                        /* Draw window */

color(WINDOW);
```

```
rectf(2.0,4.0,5.0,7.0);
rectf(9.0,4.0,12.0,7.0);

                        /* Draw window frames */

color(FRAME);
linewidth(4);
move(2.0,4.0,0.0);
draw(5.0,4.0,0.0);
draw(5.0,7.0,0.0);
draw(2.0,7.0,0.0);
draw(2.0,4.0,0.0);
move(3.5,4.0,0.0);
draw(3.5,7.0,0.0);
move(2.0,5.5,0.0);
draw(5.0,5.5,0.0);

move(9.0,4.0,0.0);
draw(12.0,4.0,0.0);
draw(12.0,7.0,0.0);
draw(9.0,7.0,0.0);
draw(9.0,4.0,0.0);
move(10.5,4.0,0.0);
draw(10.5,7.0,0.0);
move(9.0,5.5,0.0);
draw(12.0,5.5,0.0);

                        /* Draw chimney front wall */

color(SIDEWALL1);
rectf(1.0,12.0,3.0,14.2);
                        /* Draw the hole on the chimney */

color(BLACK);
rectf(1.5,13.3,2.5,13.8);

                        /* Draw top and side walls of the chimney */

chmwall1[0][0]=0.5;
chmwall1[0][1]=12.5;
chmwall1[1][0]=1.0;
chmwall1[1][1]=12.0;
chmwall1[2][0]=1.0;
```

```
chmwall1[2][1]=14.2;
chmwall1[3][0]=0.5;
chmwall1[3][1]=14.7;

color(CHMWALL1);
polf2(4,chmwall1);

chmwall2[0][0]=2.5;
chmwall2[0][1]=14.7;
chmwall2[1][0]=3.0;
chmwall2[1][1]=14.2;
chmwall2[2][0]=1.0;
chmwall2[2][1]=14.2;
chmwall2[3][0]=0.5;
chmwall2[3][1]=14.7;

color(CHMWALL2);
polf2(4,chmwall2);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makehouse1 */

/*****************************************************************

                        S P E E D L I M I T

*****************************************************************/

speedlimit(number1, number2)
char number1, number2;
{
float vertice [5][3];
Dimension legwidth = 0.2;
Dimension height = 3.3;
Dimension width = 1.625;

Dimension temp = legwidth/2.0;
Dimension temp1 = width/2.0;

                    /*  make the sign face  */
```

```
color(WHITE);
rectf(-temp1, height, temp1, height + (1.25 * width));

vertice [0][0] = - temp1 ;
vertice [0][1] = height;
vertice [0][2] = 0.0;
vertice [1][0] = - temp1 ;
vertice [1][1] = height + (1.25 * width);
vertice [1][2] = 0.0;

vertice [2][0] = temp1 ;
vertice [2][1] = height + (1.25 * width);
vertice [2][2] = 0.0;

vertice [3][0] = temp1 ;
vertice [3][1] = height;
vertice [3][2] = 0.0;

vertice [4][0] = - temp1 ;
vertice [4][1] = height;
vertice [4][2] = 0.0;

                /* put the black edge on the sign  */

color(BLACK);
linewidth(2);
poly(5, vertice);

                /* Add the speed  */

pushmatrix();
translate(-0.4, 4.0, 0.0);
scale(0.8,0.8,1.0);
translate(-5.0, -3.8, 0.0);
letter(number1, WHITE);
popmatrix();

color(BLACK);
pushmatrix();
translate(0.35, 4.0, 0.0);
scale(0.8,0.8,1.0);
translate(-5.0, -3.8, 0.0);
letter(number2, WHITE);
```

```
popmatrix();

                    /* make the leg  */

color(GRAY);
rectf(-temp, 0.0, temp, height);
} /* makethespeedlimit  */

/*****************************************************************

                        B I L L B O A R D

*****************************************************************/
billboard(num1, num2)
char num1, num2;
{
float vertice [5][3];
Dimension width = 1.625;
Dimension temp1 = width/2.0;

                    /*  make the sign face  */

color(WHITE);
rectf(-temp1, 0.0, temp1, width);

vertice [0][0] = 0.0;
vertice [0][1] = 0.0;
vertice [0][2] = 0.0;

vertice [1][0] = temp1 ;
vertice [1][1] = 0.0;
vertice [1][2] = 0.0;

vertice [2][0] = temp1 ;
vertice [2][1] = width;
vertice [2][2] = 0.0;

vertice [3][0] = - temp1 ;
vertice [3][1] = width;
vertice [3][2] = 0.0;

vertice [4][0] = - temp1 ;
vertice [4][1] = 0.0;
```

```
vertice [4][2] = 0.0;

                /* put the black edge on the sign  */

color(BLACK);
linewidth(2);
poly(5, vertice);
linewidth(1);

                /* Add the distance  */

color(RED);
pushmatrix();
translate(-0.4, 0.8, 0.0);
scale(0.8,0.8,1.0);
translate(-5.0, -3.8, 0.0);
letter(num1, WHITE);
popmatrix();

color(RED);
pushmatrix();
translate(0.35, 0.8, 0.0);
scale(0.8,0.8,1.0);
translate(-5.0, -3.8, 0.0);
letter(num2, WHITE);
popmatrix();
} /* billboard  */

/*********************************************

filename: FIND_SUBGOAL.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"

find_subgoal(no_coord, where, tolerance, pred_distance, vx, vy, px, pz, vz)
float pred_distance;
float tolerance;
```

```
float vx, vy, vz, px, pz;
int no_coord, where;
{
float dist, temp;
float x, y, z;
int i;

if (where > (no_coord - 3)) where = 0;
for (i = where; i < no_coord; ++i)
     {
     x = roadmap[i][0] - vx;
     y = roadmap[i][1] - vy;
     z = roadmap[i][2] - vz;
     dist = sqrt(x*x + y*y);
     temp = pred_distance - dist;

                    /*  converts negative to positive */
     if (temp < 0) temp = -(temp);

     if (temp <= tolerance)
          {
          if (!start)
               {
               if (vy > pz && roadmap[i][1] > vy)
                    {
                    start = TRUE;
                    return(i);
                    }
                    else if (vy < pz && roadmap[i][1] < vy)
                         {
                         start = TRUE;
                         return(i);
                         }
               }
               else
                    {
                    start = TRUE;
                    return(i);
                    }
          }
     }

                    /* If no points found, return an error code */
```

```
    return(-1);

}  /* find_subgoal */

/*********************************************

filename: INTEGRATE.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

        /* Runge-Kutta 2nd order numerical integration routine */

#include "const.h"
#include "vars.ext.h"

compute_new_state(condition)

int condition;
{
float xcap[5],. xdot[5];
int i;

derivative(state_vector, xdot, condition);
for (i = 1; i <= SYSTEM_ORDER; ++i)

                    /*  Euler prediction  */

    xcap[i] = state_vector[i] + xdot[i] * deltat;

    car_time = car_time + deltat;
    derivative(xcap, xdot, condition);
    for (i = 1; i <= SYSTEM_ORDER; ++i)

                    /* Trapezodial correction */

    state_vector[i] = (state_vector[i] + xdot[i]
                        * deltat + xcap[i])/2.0;

}  /* compute_new_state */
```

```
derivative(work_vector, xdot, condition)
float work_vector[], xdot[];
int condition;
{
xdot[1] = cos(work_vector[4]) * work_vector[3];
xdot[2] = sin(work_vector[4]) * work_vector[3];

xdot[3] = - (1/velocity_time_consant) * work_vector[3]
              + (1/velocity_time_consant) * speed;
if (condition == AUTOPILOT || condition == ASteerDrSp ||
        condition == ASteerNSp)
    {
    xdot[4] = (heading_angle_rate_gain * sigma_dot) +
               (heading_angle_gain * (sigma - work_vector[4]));
    steer_wheel_angle = xdot[4]/(turning_response_gain * work_vector[3]);
    }
    else
        {
        xdot[4] = turning_response_gain * work_vector[3]
                    * steer_wheel_angle;
        }
}  /* derivative */

/*********************************************

filename: DISPLAY.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

#include "const.h"
#include "vars.ext.h"

/************************************************************

                    S P E E D O M E T E R

************************************************************/

makethespeedometer(speedometer)
Object *speedometer;
```

```
{
     coord charxpos, pos1, pos2, tempx, tempy;
Object meter,meternum;

pos1 = 467; pos2 = 150;
tempx = pos1 + 90;
tempy = pos2 + 80;
charxpos = pos1 + 30;

/*  Generate outline for speedometer dial  */

meter=genobj();

makeobj(meter);
color(BLACK);

rectfi(pos1, pos2, tempx, tempy);

color(WHITE);
rectfi(pos1+10, pos2+10, tempx-10, tempy-10);
color(BLACK);


cmov2i(pos1,pos2-15);
charstr("  km/hr  ");

latri[0][0]=pos1;
latri[0][1]=190-9;

latri[1][0]=pos1+25;
latri[1][1]=190;

latri[2][0]=pos1;
latri[2][1]=190+9;

polf2(3,latri);

ratri[0][0]=tempx;
ratri[0][1]=190-9;

ratri[1][0]=tempx;
ratri[1][1]=190+9;
```

```
ratri[2][0]=tempx-25;
ratri[2][1]=190;
polf2(3,ratri);

closeobj();

                    /*   Generate number in speedometer display  */

meternum=genobj();

makeobj(meternum);

color(BLACK);

cmov2i(charxpos,000);
charstr("000");
cmov2i(charxpos,030);
charstr("010");
cmov2i(charxpos,060);
charstr("020");
cmov2i(charxpos,090);
charstr("030");
cmov2i(charxpos,100);
charstr("040");
cmov2i(charxpos,125);
charstr("050");
cmov2i(charxpos,150);
charstr("060");
cmov2i(charxpos,175);
charstr("070");
cmov2i(charxpos,200);
charstr("080");
cmov2i(charxpos,225);
charstr("090");
cmov2i(charxpos,250);
charstr("100");
cmov2i(charxpos,275);
charstr("110");
cmov2i(charxpos,300);
charstr("120");
cmov2i(charxpos,325);
charstr("130");
cmov2i(charxpos,350);
```

```
charstr("140");
cmov2i(charxpos,375);
charstr("150");
cmov2i(charxpos,400);
charstr("160");
cmov2i(charxpos,425);
charstr("170");
cmov2i(charxpos,450);
charstr("180");
cmov2i(charxpos,475);
charstr("190");

closeobj();

                /*   Put all pieces of speedometer  together   */

*speedometer=genobj();
makeobj(*speedometer);

                /* Draw the boundary  */

callobj(meter);

                /*  Draw the display speedometer in the  window */

scrmask(charxpos,tempx,pos2+10,tempy-10);

pushmatrix();
transl4=gentag();
maketag(transl4);
translate(0.0,0.0,0.0);
callobj(meternum);
popmatrix();

                /*  Reset screenmask to full size screen  */
scrmask(0,1023,0,767);

viewport(0,1023,0,767);

closeobj();

}  /*  makethespeedometer  */
```

```
/*************************************************************

                        F U E L    M E T E R

*************************************************************/

makefuel(fuel)
Object *fuel;
{
Coord fuelx1, fuelx2, fuely1, fuely2;
Object fuelbound,fuellevel;

fuelx1 = 277.0; fuelx2 = fuelx1 + 51.0;
fuely1 = 10.0; fuely2 = 340.0;
                    /* Generate outline for fuel indicator */

fuelbound=genobj();
makeobj(fuelbound);

color(BLACK);
rectf(fuelx1, fuely1, fuelx2, fuely2);

cmov2(fuelx1 + 5.0,345.0);
charstr("fuel");

                    /* Generate hash marks for fuel levels */

linewidth(3);

move(fuelx2, fuely2-30.0, 0.0);
rdr(5.0, 0.0, 0.0);

move(fuelx2, fuely1+60.0, 0.0);
rdr(5.0, 0.0, 0.0);

linewidth(1);

closeobj();

                    /* Generate the fuel level bar that moves */

fuellevel=genobj();
makeobj(fuellevel);
```

```
color(WHITE);
rectf(fuelx1+4.0, fuely1+4.0, fuelx2-4.0, fuely2-4.0);

closeobj();

                    /* Put all pieces of fuel together */

*fuel=genobj();
makeobj(*fuel);
callobj(fuelbound);

callobj(fuellevel);
color(YELLOW);

fuel1 = gentag();
maketag(fuel1);
rectf(fuelx1+4.0, fuely1+4.0, fuelx2-4.0, fuely2-4.0);
color(BLACK);

closeobj();

} /* makefuel */

/*************************************************************

                        H E L P     P A N E L

*************************************************************/

makehelp(help)
Object *help;
{
*help=genobj();
makeobj(*help);

color(BLACK);

cmov2i(102, 345);
charstr("controls");

linewidth(5);
```

```
recti(10,10,265,340);

                /* Generate info on display */

cmov2i(30, 315);
charstr("Q: AutoPilot");

cmov2i(30, 295);
charstr("C: Driver Steer");

cmov2i(30, 275);
charstr("R: Cruise, Nav Steer");

cmov2i(30, 255);
charstr("S: AutoSt, Dr Speed");

cmov2i(30, 235);
charstr("A: AutoSt, Nav Speed");

cmov2i(30, 215);
charstr("D: Driver Control");

cmov2i(30, 195);
charstr("W: Navigator Control");

cmov2i(30, 175);
charstr("X: Dr Steers, Nav's Sp");

cmov2i(30, 155);
charstr("F: Nav Steers, Dr's Sp");

cmov2i(30, 135);
charstr("E: Exit");

linewidth(2);

cmov2i(28,95);
charstr("Speed");
circi(29,80,6);
circi(50,80,6);
color(RED);
circfi(71,80,6);
```

```
color(BLACK);
cmov2i(28.55);
charstr(" Decel");
color(BLACK);
circi(29,40,6);
color(RED);
circfi(50,40,6);
color(BLACK);
circi(71,40,6);

cmov2i(140, 95);
charstr("Steering");

move2i(140, 80);
draw2i(185, 80);

move2i(150, 85);
draw2i(140, 80);

move2i(150, 75);
draw2i(140, 80);

move2i(175, 75);
draw2i(185, 80);

move2i(175, 85);
draw2i(185, 80);

cmov2i(140, 55);
charstr("Brakes");

move2i(215, 65);
draw2i(215, 25);

move2i(210, 55);
draw2i(215, 65);

move2i(220, 55);
draw2i(215, 65);

move2i(210, 35);
draw2i(215, 25);
```

```
move2i(220, 35);
draw2i(215, 25);

linewidth(1);

closeobj();

}  /*  makehelp  */

/*****************************************************************

                        O D O M E T E R

*****************************************************************/

maketheodometer(odometer)
Object *odometer;
{
     coord pos1, pos2, tempx, tempy;
Coord temp, charx, chary;
pos1 = 467; pos2 = 50;
tempx = pos1 + 90; tempy = pos2 + 50;

*odometer = genobj();
makeobj(*odometer);
color(BLACK);
rectfi(pos1, pos2, tempx, tempy);
color(WHITE);
rectfi(pos1+5, pos2+5, tempx-5, tempy-5);
color(BLACK);

temp = (tempx - pos1 - 10)/4;
move2(pos1+5+temp, pos2+5);
draw2(pos1+5+temp, tempy-5);

move2(pos1+5+temp*2, pos2+5);
draw2(pos1+5+temp*2, tempy-5);

move2(pos1+5+temp*3, pos2+5);
draw2(pos1+5+temp*3, tempy-5);

move2(pos1+5+temp*4, pos2+5);
draw2(pos1+5+temp*4, tempy-5);
```

```
charx = pos1+5+temp/2; chary = (tempy-pos2)/2+pos2-5.0;

cmov2(charx, chary);
odotag1 = gentag();
maketag(odotag1);
charstr("0");

cmov2(charx + temp, chary);
odotag2 = gentag();
maketag(odotag2);
charstr("0");

cmov2(charx + temp*2, chary);
odotag3 = gentag();
maketag(odotag3);
charstr("0");

cmov2(charx + temp*3, chary);
odotag4 = gentag();
maketag(odotag4);
charstr("0");

color(BLACK);
cmov2i(pos1,pos2-15);
charstr("    meter");
closeobj();
} /* maketheodometer */


/*************************************************************

                    W A R N I N G      P A N E L

*************************************************************/

makewarning(warning)
Object *warning;
{
Coord tempx, tempy, pos1, pos2;
Coord ix, iy, tempy1, tempy2, tempy3, tempy4, hg;

pos1 = 840.0; pos2 = 10.0;
tempx = pos1 + 140.0; tempy = 340.0;
```

```
iy = ix = 20.0;
*warning = genobj();
makeobj(*warning);

color(BLACK);
rectf(pos1, pos2, tempx, tempy);
hg = (330 - 5*iy)/4;

dangertag = gentag();
maketag(dangertag);
color(RED);
rectf(pos1+ix, pos2+iy, tempx-ix, pos2+iy+hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 25.0, pos2+iy+hg/2-5.0);
charstr("Danger");

temptag = gentag();
maketag(temptag);
color(RED);
rectf(pos1+ix,pos2+iy*2+hg,tempx-ix,pos2+iy*2+2*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 12.0, pos2+iy*2+hg+hg/2-5.0);
charstr("Temp");

belttag = gentag();
maketag(belttag);
color(RED);
rectf(pos1+ix,pos2+iy*3+hg*2,tempx-ix,pos2+iy*3+3*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 40.0, pos2+iy*3+hg*2+hg/2-5.0);
charstr("Seat Belt");

braketag = gentag();
maketag(braketag);
color(RED);
rectf(pos1+ix,pos2+iy*4+hg*3,tempx-ix,pos2+iy*4+4*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 17.0, pos2+iy*4+hg*3+hg/2-5.0);
charstr("Brake");

color(BLACK);
cmov2(pos1+20.0, tempy+5.0);
charstr("  Warning");
```

```
closeobj();
}  /* makewarning */


/*************************************************************

                    S T E E R I N G     W H E E L

*************************************************************/

makesteerwheel(steerwheel)
Object *steerwheel;
{
*steerwheel = genobj();
makeobj(*steerwheel);

pushmatrix();
color(BLACK);
circfi(512, 290, 40);
color(WHITE);
circfi(512, 290, 30);
color(BLACK);

translate(512.0, 290.0, 0.0);
steerwheeltag = gentag();
maketag(steerwheeltag);
rotate(0, 'Z');
rectfi(-33, -5, 33, 5);

popmatrix();
closeobj();

}  /* makesteerwheel */


/*************************************************************

                    H E A D I N G     M E T E R

*************************************************************/

makeheading(heading_meter)
Object *heading_meter;
```

```
{
Object meter, theading;
Coord pos1, pos2, tempx, tempy;
pos1 = heading_xpos; pos2 = 350.0;
tempx = pos1 + 175.0;
tempy = pos2 + 12.5;

meter = genobj();
makeobj(meter);
color(BLACK);
rectf(pos1-2.5, pos2-2.5, tempx+2.5, tempy+3.5);
color(WHITE);
rectf(pos1, pos2, tempx, tempy + 1.0);
closeobj();

                /*  Generate the heading on top of the terrain map  */

theading=genobj();
makeobj(theading);
color(BLACK);

cmov2(000.0,pos2-2.0);
charstr("340");

cmov2(045.0,pos2-2.0);
charstr("350");

cmov2(090.0,pos2-2.0);
charstr("360");

cmov2(135.0,pos2-2.0);
charstr("010");

cmov2(180.0,pos2-2.0);
charstr("020");

cmov2(225.0,pos2-2.0);
charstr("030");

cmov2(270.0,pos2-2.0);
charstr("040");

cmov2(315.0,pos2-2.0);
```

```
charstr("050");

cmov2(360.0,pos2-2.0);
charstr("060");

cmov2(405.0,pos2-2.0);
charstr("070");

cmov2(450.0,pos2-2.0);
charstr("080");

cmov2(495.0,pos2-2.0);
charstr("090");

cmov2(540.0,pos2-2.0);
charstr("100");

cmov2(585.0,pos2-2.0);
charstr("110");

cmov2(630.0,pos2-2.0);
charstr("120");

cmov2(675.0,pos2-2.0);
charstr("130");

cmov2(720.0,pos2-2.0);
charstr("140");

cmov2(765.0,pos2-2.0);
charstr("150");

cmov2(810.0,pos2-2.0);
charstr("160");

cmov2(855.0,pos2-2.0);
charstr("170");

cmov2(900.0,pos2-2.0);
charstr("180");

cmov2(945.0,pos2-2.0);
charstr("190");
```

```
cmov2(990.0,pos2-2.0);
charstr("200");

cmov2(1035.0,pos2-2.0);
charstr("210");

cmov2(1080.0,pos2-2.0);
charstr("220");

cmov2(1125.0,pos2-2.0);
charstr("230");

cmov2(1170.0,pos2-2.0);
charstr("240");

cmov2(1215.0,pos2-2.0);
charstr("250");

cmov2(1260.0,pos2-2.0);
charstr("260");

cmov2(1305.0,pos2-2.0);
charstr("270");

cmov2(1350.0,pos2-2.0);
charstr("280");

cmov2(1395.0,pos2-2.0);
charstr("290");

cmov2(1440.0,pos2-2.0);
charstr("300");

cmov2(1485.0,pos2-2.0);
charstr("310");

cmov2(1530.0,pos2-2.0);
charstr("320");

cmov2(1575.0,pos2-2.0);
charstr("330");

cmov2(1620.0,pos2-2.0);
```

```
charstr("340");

cmov2(1665.0,pos2-2.0);
charstr("350");

cmov2(1710.0,pos2-2.0);
charstr("360");

cmov2(1755.0,pos2-2.0);
charstr("010");

cmov2(1800.0,pos2-2.0);
charstr("020");

color(BLACK);

closeobj();

                    /*  Put all the pieces together  */
*heading_meter=genobj();
makeobj(*heading_meter);

                    /*  Draw the boundary */
callobj(meter);
                    /*  Draw the heading */

scrmask((int) pos1,(int) tempx,(int) pos2,(int) tempy);

pushmatrix();
transl1=gentag();
maketag(transl1);

translate(0.0,0.0,0.0);

callobj(theading);
scrmask(0,1023,0,767);
popmatrix();

color(RED);
linewidth(4);
move2(pos1+175.0/2,pos2);
draw2(pos1+175.0/2,tempy);
```

```
linewidth(1);

scrmask(0,1023,0,767);
closeobj();

}  /* makeheading */

/*************************************************************

                    B R A K E / C M D S P E E D G U A G E S

*************************************************************/

makegauges(gauges)
Object *gauges;
{

*gauges = genobj();
makeobj(*gauges);

                    /* make the brake gauge  */

cmov2i(BRAKEX + 4, BRAKEY + 210);
charstr("brakes");

color(RED);
manbraketag = gentag();
maketag(manbraketag);
rectfi(BRAKEX, BRAKEY, BRAKEX + 50, BRAKEY);
color(BLACK);
scalegauge(BRAKEX, BRAKEY);

                    /* make the cmdspeed gauge  */

cmov2i(CMDX - 6, CMDY + 226);
charstr("command");

cmov2i(CMDX + 2, CMDY + 210);
charstr("speed");

color(GREEN);
manspeedtag = gentag();
maketag(manspeedtag);
```

```
rectfi(CMDX, CMDY, CMDX + 50, CMDY);
scalegauge(CMDX, CMDY);

closeobj();
}

/*************************************************************

                    S C A L E G U A G E ();

*************************************************************/

scalegauge(basex, basey)
int basex, basey;
{
char tempstr[10];
int i;

                    /* outline the gauge */

linewidth(2);
color(BLACK);
recti(basex, basey, basex + 50, basey + 200);
linewidth(1);

                    /* calibrate the gauge */

for (i = 10; i < 100; i = i + 10)
    {
    move2i(basex, basey + 2 * i);
    draw2i(basex + 13, basey + 2 * i);
    move2i(basex + 37, basey + 2 * i);
    draw2i(basex + 50, basey + 2 * i);
    cmov2i(basex + 16, basey + (2 * i) - 4);
    sprintf(tempstr, "%d", i);
    charstr(tempstr);
    }
}   /*  scalegauge()  */

/*********************************************
```

filename: CHECKKEY.C
author: Michael J. Dolezal

```
date: May 20, 1987

*********************************************/

#include "gl.h"
#include "const.h"
#include "device.h"
#include "vars.ext.h"

checkkeybd(ptnotdone, ptdebug, ptstart, ptmode, ptcondition)

Boolean *ptnotdone, *ptdebug, *ptstart;
int *ptmode, *ptcondition;
{

keypressed = NULL;

*ptmode = *ptcondition;

if (qtest())
    {
    qread(&keypressed);

    switch(keypressed)
        {
        case 'q':
        case 'Q': if (state_vector[3] > 3.0)
                {
                *ptmode = AUTOPILOT;
                *ptstart = FALSE;
                }
                break;
                                /* Cruise cont and driver steer */
        case 'c':
        case 'C': if ((*ptmode == ASteerNSp || *ptmode == ASteerDrSp) &&
                        (state_vector[3] > 3.0))
                {
                *ptmode = AUTOPILOT;
                *ptstart = FALSE;
                }
                else *ptmode = CruiseDrSteer;
                break;
                                /* Cruise cont and remote steer  */
```

```
case 'r':
case 'R': if (*ptmode == ASteerNSp || *ptmode == ASteerDrSp)
      {
      *ptmode = AUTOPILOT;
      *ptstart = FALSE;
      }
      else *ptmode = CruiseNavSteer;
      break;
                        /* Auto speed and driver speed */
case 's':
case 'S': if (state_vector[3] > 3.0)
      {
      if (*ptmode == CruiseNavSteer || *ptmode == CruiseDrSteer)
           {
           *ptmode = AUTOPILOT;
           *ptstart = FALSE;
           }
           else *ptmode = ASteerDrSp;
      }
      break;
                        /* Auto speed and remote steer  */
case 'a':
case 'A': if (state_vector[3] > 3.0)
      {
      if (*ptmode == CruiseNavSteer || *ptmode == CruiseDrSteer)
          {
          *ptmode = AUTOPILOT;
          *ptstart = FALSE;
          }
          else *ptmode = ASteerNSp;
      }
      break;
                        /* All remote manual control */
case 'w':
case 'W': *ptmode = NavManual;
          *ptstart = FALSE:
          break;

case 'd':
case 'D': *ptmode = DrManual;
          *ptstart = FALSE;
          break;
```

```
        case 'x':
        case 'X': *ptmode = DrSteerNavSp;
                  *ptstart = FALSE;
                  break;

        case 'f':
        case 'F': *ptmode = NavSteerDrSp;
                  *ptstart = FALSE;
                  break;

        case 'e':
        case 'E': *ptnotdone = FALSE;
                  break;

       }
    }
*ptcondition = *ptmode;

}  /* checkkeybd */

/***********************************************

filename: WELCOME.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

***********************************************/
```

This same module is used on the Navigator's Display
and can be found in Appendix A of this study.

```
/***********************************************

filename: LETTER.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

***********************************************/

/* This file contains routines to display block alphabetic characters
   suitable for inclusion into graphics objects.  These letters are
```

```
   used instead of IRIS FONTS when one desires to treat them as
   graphics objects that can be rotated, scaled, etc. (font char-
   acters can't)                                                    */

/* This file includes routines for 27 characters, "A" through "Z",
   and also ":" and " " (blank) (but not "G","Q","V","W","X",Z")    */

/* The routine draws the desired letter in absolute coordinates.
   in the center of the display.                                    */

/* To use these routines, the color desired for the letter must
   be specified when the object is created (in the user program),
   and the desired backgound color must be passed to the routine.   */

/* Original version written by J. Artero and R. Kirsch; current
   version written by L. Williamson, digits added by Mike Dolezal.  */


#include "gl.h"
#include "device.h"

letter(asci,backcolor)

int asci;                   /* index of character we want to display  */

Colorindex backcolor;       /* specified background color             */

{

   Coord box [8][2];        /* vector of coordinates forming the
                               vertices of a letter object            */


   switch(asci)
   {

      case 'A':

         box[0][0]=4.6875;
         box[0][1]=3.25;
         box[1][0]=4.9375;
         box[1][1]=4.25;
         box[2][0]=5.0625;
```

```
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=3.25;
        polf2(4,box);

        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.25;
        box[1][0]=4.84375;
        box[1][1]=3.375;
        box[2][0]=5.15625;
        box[2][1]=3.375;
        box[3][0]=5.1875;
        box[3][1]=3.25;
        polf2(4,box);
        box[0][0]=4.875;
        box[0][1]=3.5;
        box[1][0]=5.0;
        box[1][1]=4.0;
        box[2][0]=5.125;
        box[2][1]=3.5;
        polf2(3,box);

        break;

    case 'B':

        box[0][0]=4.6875;
        box[0][1]=3.25;
        box[1][0]=4.6875;
        box[1][1]=4.25;
        box[2][0]=5.1875;
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=4.125;
        box[4][0]=5.3125;
        box[4][1]=3.375;
        box[5][0]=5.1875;
        box[5][1]=3.25;
        polf2(6,box);

        color(backcolor);
        box[0][0]=5.25;
```

```
    box[0][1]=3.8125;
    box[1][0]=5.3125;
    box[1][1]=3.875;
    box[2][0]=5.3125;
    box[2][1]=3.75;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.375;
    box[1][0]=4.8125;
    box[1][1]=3.75;
    box[2][0]=5.125;
    box[2][1]=3.75;
    box[3][0]=5.1875;
    box[3][1]=3.6875;
    box[4][0]=5.1875;
    box[4][1]=3.4375;
    box[5][0]=5.125;
    box[5][1]=3.375;
    polf2(6,box);

    box[0][0]=4.8125;
    box[0][1]=3.875;
    box[1][0]=4.8125;
    box[1][1]=4.125;
    box[2][0]=5.125;
    box[2][1]=4.125;
    box[3][0]=5.1875;
    box[3][1]=4.0625;
    box[4][0]=5.1875;
    box[4][1]=3.9375;
    box[5][0]=5.125;
    box[5][1]=3.875;
    polf2(6,box);
    break;
case 'C':

    box[0][0]=4.6875;
    box[0][1]=3.375;
    box[1][0]=4.6875;
    box[1][1]=4.125;
    box[2][0]=4.8125;
    box[2][1]=4.25;
```

```
    box[3][0]=5.1875;
    box[3][1]=4.25;
    box[4][0]=5.3125;
    box[4][1]=4.125;
    box[5][0]=5.3125;
    box[5][1]=3.375;
    box[6][0]=5.1875;
    box[6][1]=3.25;
    box[7][0]=4.8125;
    box[7][1]=3.25;
    polf2(8,box);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.4375;
    box[1][0]=4.8125;
    box[1][1]=4.0625;
    box[2][0]=4.875;
    box[2][1]=4.125;
    box[3][0]=5.125;
    box[3][1]=4.125;
    box[4][0]=5.1875;
    box[4][1]=4.0625;
    box[5][0]=5.1875;
    box[5][1]=3.4375;
    box[6][0]=5.125;
    box[6][1]=3.375;
    box[7][0]=4.875;
    box[7][1]=3.375;
    polf2(8,box);

    rectf(5.1875,3.5,5.3125,4.00);

    break;

case 'D':

    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=4.25;
```

```
    box[3][0]=5.3125;
    box[3][1]=4.125;
    box[4][0]=5.3125;
    box[4][1]=3.375;
    box[5][0]=5.1875;
    box[5][1]=3.25;
    polf2(6,box);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.375;
    box[1][0]=4.8125;
    box[1][1]=4.125;
    box[2][0]=5.125;
    box[2][1]=4.125;
    box[3][0]=5.1875;
    box[3][1]=4.0625;
    box[4][0]=5.1875;
    box[4][1]=3.4375;
    box[5][0]=5.125;
    box[5][1]=3.375;
    polf2(6,box);

    break;

case 'E':

    rectf(4.6875,4.125,5.25,4.25);
    rectf(4.6875,3.25,5.3125,3.375);
    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.8125,3.75,5.0625,3.875);

    break;

case 'F':

    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.6875,4.125,5.3125,4.25);
    rectf(4.8125,3.75,5.125,3.875);

    break;

case 'H':
```

```
        rectf(4.6875,3.25,4.8125,4.25);
        rectf(4.8125,3.6875,5.1875,3.8125);
        rectf(5.1875,3.25,5.3125,4.25);

        break;

    case 'I':

        rectf(4.6875,4.125,5.3125,4.25);
        rectf(4.6875,3.25,5.3125,3.375);
        rectf(4.9375,3.25,5.0625,4.25);

        break;

    case 'J':

        box[0][0]=4.6875;
        box[0][1]=3.375;
        box[1][0]=4.6875;
        box[1][1]=3.625;
        box[2][0]=5.3125;
        box[2][1]=3.625;
        box[3][0]=5.3125;
        box[3][1]=3.375;
        box[4][0]=5.1875;
        box[4][1]=3.25;
        box[5][0]=4.8125;
        box[5][1]=3.25;
        polf2(6,box);

        rectf(5.2,3.625,5.3125,4.25);
        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.4375;
        box[1][0]=4.8125;
        box[1][1]=3.625;
        box[2][0]=5.1875;
        box[2][1]=3.625;
        box[3][0]=5.1875;
        box[3][1]=3.4375;
        box[4][0]=5.125;
        box[4][1]=3.375;
        box[5][0]=4.875;
```

```
    box[5][1]=3.375;
    polf2(6,box);

    break;

case 'K':

    rectf(4.6875,3.25,5.3125,4.25);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.875;
    box[1][0]=4.8125;
    box[1][1]=4.25;
    box[2][0]=5.125;
    box[2][1]=4.25;
    polf2(3,box);

    box[0][0]=5.02;
    box[0][1]=3.875;
    box[1][0]=5.3125;
    box[1][1]=4.25;
    box[2][0]=5.3125;
    box[2][1]=3.25;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=4.8125;
    box[1][1]=3.625;
    box[2][0]=4.9;
    box[2][1]=3.74;
    box[3][0]=5.14;
    box[3][1]=3.25;
    polf2(4,box);

    break;

case 'L':

    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.6875,3.25,5.3125,3.375);
```

```
        break;

    case 'M':

        rectf(4.6875,3.25,5.3125,4.25);
        color(backcolor);
        box[0][0]=4.6875;
        box[0][1]=4.25;
        box[1][0]=5.3125;
        box[1][1]=4.25;
        box[2][0]=5.0;
        box[2][1]=3.75;
        polf2(3,box);

        box[0][0]=4.8125;
        box[0][1]=3.25;
        box[1][0]=4.8125;
        box[1][1]=3.8125;
        box[2][0]=5.125;
        box[2][1]=3.25;
        polf2(3,box);

        box[0][0]=4.875;
        box[0][1]=3.25;
        box[1][0]=5.1875;
        box[1][1]=3.8125;
        box[2][0]=5.1875;
        box[2][1]=3.25;
        polf2(3,box);

        break;

    case 'N':

        rectf(4.6875,3.25,5.3125,4.25);

        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.25;
        box[1][0]=4.8125;
        box[1][1]=3.9375;
        box[2][0]=5.1875;
        box[2][1]=3.25;
```

```
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=4.25;
    box[1][0]=5.1875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=3.5625;
    polf2(3,box);

    break;

case 'O':

    box[0][0]=4.6875;
    box[0][1]=3.375;
    box[1][0]=4.6875;
    box[1][1]=4.125;
    box[2][0]=4.8125;
    box[2][1]=4.25;
    box[3][0]=5.1875;
    box[3][1]=4.25;
    box[4][0]=5.3125;
    box[4][1]=4.125;
    box[5][0]=5.3125;
    box[5][1]=3.375;
    box[6][0]=5.1875;
    box[6][1]=3.25;
    box[7][0]=4.8125;
    box[7][1]=3.25;
    polf2(8,box);
    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.4375;
    box[1][0]=4.8125;
    box[1][1]=4.0625;
    box[2][0]=4.875;
    box[2][1]=4.125;
    box[3][0]=5.125;
    box[3][1]=4.125;
    box[4][0]=5.1875;
    box[4][1]=4.0625;
    box[5][0]=5.1875;
```

```
        box[5][1]=3.4375;
        box[6][0]=5.125;
        box[6][1]=3.375;
        box[7][0]=4.875;
        box[7][1]=3.375;
        polf2(8,box);
        break;

    case 'P':

        box[0][0]=4.6875;
        box[0][1]=3.25;
        box[1][0]=4.6875;
        box[1][1]=4.25;
        box[2][0]=5.1875;
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=4.125;
        box[4][0]=5.3125;
        box[4][1]=3.25;
        polf2(5,box);

        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.25;
        box[1][0]=5.3125;
        box[1][1]=3.8125;
        box[2][0]=5.3125;
        box[2][1]=3.25;
        polf2(3,box);

        box[0][0]=4.8125;
        box[0][1]=3.8125;
        box[1][0]=4.8125;
        box[1][1]=4.125;
        box[2][0]=5.125;
        box[2][1]=4.125;
        box[3][0]=5.1875;
        box[3][1]=4.0625;
        box[4][0]=5.1875;
        box[4][1]=3.875;
        box[5][0]=5.125;
        box[5][1]=3.8125;
```

```
        polf2(6,box);
        rectf(4.8125,3.25,5.3125,3.6875);

        break;

    case 'R':

        box[0][0]=4.6875;
        box[0][1]=3.25;
        box[1][0]=4.6875;
        box[1][1]=4.25;
        box[2][0]=5.1875;
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=4.125;
        box[4][0]=5.3125;
        box[4][1]=3.25;
        polf2(5,box);

        color(backcolor);
        box[0][0]=5.1875;
        box[0][1]=3.625;
        box[1][0]=5.3125;
        box[1][1]=3.75;
        box[2][0]=5.3125;
        box[2][1]=3.25;
        polf2(3,box);

        box[0][0]=4.8125;
        box[0][1]=3.75;
        box[1][0]=4.8125;
        box[1][1]=4.125;
        box[2][0]=5.125;
        box[2][1]=4.125;
        box[3][0]=5.1875;
        box[3][1]=4.0625;
        box[4][0]=5.1875;
        box[4][1]=3.8125;
        box[5][0]=5.125;
        box[5][1]=3.75;
        polf2(6,box);

        box[0][0]=4.8125;
```

```
        box[0][1]=3.25;
        box[1][0]=4.8125;
        box[1][1]=3.625;
        box[2][0]=5.05;
        box[2][1]=3.625;
        box[3][0]=5.175;
        box[3][1]=3.25;
        polf2(4,box);

        break;

    case 'S':

        box[0][0]=4.6875;
        box[0][1]=3.375;
        box[1][0]=4.6875;
        box[1][1]=4.125;
        box[2][0]=4.8125;
        box[2][1]=4.25;
        box[3][0]=5.1875;
        box[3][1]=4.25;
        box[4][0]=5.3125;
        box[4][1]=4.125;
        box[5][0]=5.3125;
        box[5][1]=3.375;
        box[6][0]=5.1875;
        box[6][1]=3.25;
        box[7][0]=4.8125;
        box[7][1]=3.25;
        polf2(8,box);

        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.4375;
        box[1][0]=4.8125;
        box[1][1]=3.75;
        box[2][0]=5.125;
        box[2][1]=3.75;
        box[3][0]=5.1875;
        box[3][1]=3.6875;
        box[4][0]=5.1875;
        box[4][1]=3.4375;
        box[5][0]=5.125;
```

```
    box[5][1]=3.375;
    box[6][0]=4.875;
    box[6][1]=3.375;
    polf2(7,box);

    box[0][0]=4.8125;
    box[0][1]=3.9375;
    box[1][0]=4.8125;
    box[1][1]=4.0625;
    box[2][0]=4.875;
    box[2][1]=4.125;
    box[3][0]=5.125;
    box[3][1]=4.125;
    box[4][0]=5.1875;
    box[4][1]=4.0625;
    box[5][0]=5.1875;
    box[5][1]=3.875;
    box[6][0]=4.875;
    box[6][1]=3.875;
    polf2(7,box);

    box[0][0]=4.6875;
    box[0][1]=3.5625;
    box[1][0]=4.6875;
    box[1][1]=3.875;
    box[2][0]=4.8125;
    box[2][1]=3.75;
    box[3][0]=4.8125;
    box[3][1]=3.5625;
    polf2(4,box);

    box[0][0]=5.1875;
    box[0][1]=3.875;
    box[1][0]=5.1875;
    box[1][1]=4.0;
    box[2][0]=5.3125;
    box[2][1]=4.0;
    box[3][0]=5.3125;
    box[3][1]=3.75;
    polf2(4,box);
    break;

case 'T':
```

```
        rectf(4.6875,4.125,5.3125,4.25);
        rectf(4.9375,3.25,5.0625,4.25);
        break;

    case 'U':

        box[0][0]=4.6875;
        box[0][1]=3.375;
        box[1][0]=4.6875;
        box[1][1]=4.25;
        box[2][0]=5.3125;
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=3.25;
        box[4][0]=4.8125;
        box[4][1]=3.25;
        polf2(5,box);

        color(backcolor);
        box[0][0]=4.8125;
        box[0][1]=3.4375;
        box[1][0]=4.8125;
        box[1][1]=4.25;
        box[2][0]=5.1875;
        box[2][1]=4.25;
        box[3][0]=5.1875;
        box[3][1]=3.5325;
        box[4][0]=5.01;
        box[4][1]=3.375;
        box[5][0]=4.875;
        box[5][1]=3.375;
        polf2(6,box);

        box[0][0]=5.0625;
        box[0][1]=3.25;
        box[1][0]=5.1875;
        box[1][1]=3.375;
        box[2][0]=5.1875;
        box[2][1]=3.25;
        polf2(3,box);

        break;
```

```
case 'Y':

    box[0][0]=4.6875;
    box[0][1]=4.25;
    box[1][0]=4.9375;
    box[1][1]=3.75;
    box[2][0]=5.0625;
    box[2][1]=3.75;
    box[3][0]=4.8125;
    box[3][1]=4.25;
    polf2(4,box);
    box[0][0]=4.9375;
    box[0][1]=3.75;
    box[1][0]=5.0625;
    box[1][1]=3.75;
    box[2][0]=5.3125;
    box[2][1]=4.25;
    box[3][0]=5.1875;
    box[3][1]=4.25;
    polf2(4,box);

    rectf(4.9375,3.25,5.0625,3.75);

    break;

case '1':

    rectf(4.9375, 3.25, 5.0625, 4.25);

    break;

case '2':

    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=4.125;
    box[4][0]=5.3125;
    box[4][1]=3.375;
```

```
box[5][0]=5.1875;
box[5][1]=3.25;
polf2(6,box);

color(backcolor);
box[0][0]=5.25;
box[0][1]=3.8125;
box[1][0]=5.3125;
box[1][1]=3.875;
box[2][0]=5.3125;
box[2][1]=3.75;
polf2(3,box);

box[0][0]=4.8125;
box[0][1]=3.375;
box[1][0]=4.8125;
box[1][1]=3.75;
box[2][0]=5.125;
box[2][1]=3.75;
box[3][0]=5.1875;
box[3][1]=3.6875;
box[4][0]=5.1875;
box[4][1]=3.4375;
box[5][0]=5.125;
box[5][1]=3.375;
polf2(6,box);

box[0][0]=4.8125;
box[0][1]=3.875;
box[1][0]=4.8125;
box[1][1]=4.125;
box[2][0]=5.125;
box[2][1]=4.125;
box[3][0]=5.1875;
box[3][1]=4.0625;
box[4][0]=5.1875;
box[4][1]=3.9375;
box[5][0]=5.125;
box[5][1]=3.875;
polf2(6,box);

color(backcolor);
box[0][0]=4.8125;
```

```
        box[0][1]=3.875;
        box[1][0]=4.8125;
        box[1][1]=4.125;
        box[2][0]=4.6875;
        box[2][1]=4.125;
        box[3][0]=4.6875;
        box[3][1]=3.875;
        polf2(4,box);

        color(backcolor);
        box[0][0]=5.3125;
        box[0][1]=3.875;
        box[1][0]=5.125;
        box[1][1]=3.75;
        box[2][0]=5.125;
        box[2][1]=3.45;
        box[3][0]=5.3125;
        box[3][1]=3.45;
        polf2(4,box);

        break;

case '3':

        box[0][0]=4.6875;
        box[0][1]=3.25;
        box[1][0]=4.6875;
        box[1][1]=4.25;
        box[2][0]=5.1875;
        box[2][1]=4.25;
        box[3][0]=5.3125;
        box[3][1]=4.125;
        box[4][0]=5.3125;
        box[4][1]=3.375;
        box[5][0]=5.1875;
        box[5][1]=3.25;
        polf2(6,box);

        color(backcolor);
        box[0][0]=5.25;
        box[0][1]=3.8125;
        box[1][0]=5.3125;
        box[1][1]=3.875;
```

```
box[2][0]=5.3125;
box[2][1]=3.75;
polf2(3,box);
box[0][0]=4.8125;
box[0][1]=3.375;
box[1][0]=4.8125;
box[1][1]=3.75;
box[2][0]=5.125;
box[2][1]=3.75;
box[3][0]=5.1875;
box[3][1]=3.6875;
box[4][0]=5.1875;
box[4][1]=3.4375;
box[5][0]=5.125;
box[5][1]=3.375;
polf2(6,box);

box[0][0]=4.8125;
box[0][1]=3.875;
box[1][0]=4.8125;
box[1][1]=4.125;
box[2][0]=5.125;
box[2][1]=4.125;
box[3][0]=5.1875;
box[3][1]=4.0625;
box[4][0]=5.1875;
box[4][1]=3.9375;
box[5][0]=5.125;
box[5][1]=3.875;
polf2(6,box);

color(backcolor);
box[0][0]=4.6875;
box[0][1]=3.375;
box[1][0]=4.6875;
box[1][1]=4.125;
box[2][0]=4.8125;
box[2][1]=4.125;
box[3][0]=4.8125;
box[3][1]=3.375;
polf2(4,box);

break;
```

```
case '4':

    rectf(5.1875, 3.25, 5.3125, 4.25);
    rectf(4.6875, 3.75, 5.3125, 3.875);
    rectf(4.6875, 3.75, 4.8125, 4.25);

    break;

case '5':

    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.3125;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=3.375;
    box[4][0]=5.1875;
    box[4][1]=3.25;
    polf2(5,box);
    color(backcolor);
    box[0][0]=5.25;
    box[0][1]=3.8125;
    box[1][0]=5.3125;
    box[1][1]=3.875;
    box[2][0]=5.3125;
    box[2][1]=3.75;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.375;
    box[1][0]=4.8125;
    box[1][1]=3.75;
    box[2][0]=5.125;
    box[2][1]=3.75;
    box[3][0]=5.1875;
    box[3][1]=3.6875;
    box[4][0]=5.1875;
    box[4][1]=3.4375;
    box[5][0]=5.125;
    box[5][1]=3.375;
    polf2(6,box);
```

```
        box[0][0]=4.8125;
        box[0][1]=3.875;
        box[1][0]=4.8125;
        box[1][1]=4.125;
        box[2][0]=5.125;
        box[2][1]=4.125;
        box[3][0]=5.1875;
        box[3][1]=4.0625;
        box[4][0]=5.1875;
        box[4][1]=3.9375;
        box[5][0]=5.125;
        box[5][1]=3.875;
        polf2(6,box);

        color(backcolor);
        box[0][0]=5.25;
        box[0][1]=3.8125;
        box[1][0]=5.3125;
        box[1][1]=3.875;
        box[2][0]=5.3125;
        box[2][1]=4.125;
        box[3][0]=5.125;
        box[3][1]=4.125;
        box[4][0]=5.125;
        box[4][1]=3.875;
        polf2(5,box);

        box[0][0]=4.8125;
        box[0][1]=3.375;
        box[1][0]=4.8125;
        box[1][1]=3.75;
        box[2][0]=4.6875;
        box[2][1]=3.75;
        box[3][0]=4.6875;
        box[3][1]=3.375;
        polf2(4,box);

        break;

    case '6':
        box[0][0]=4.6875;
        box[0][1]=3.25;
        box[1][0]=4.6875;
```

```
box[1][1]=4.25;
box[2][0]=5.1875;
box[2][1]=4.25;
box[3][0]=5.3125;
box[3][1]=4.125;
box[4][0]=5.3125;
box[4][1]=3.375;
box[5][0]=5.1875;
box[5][1]=3.25;
polf2(6,box);

color(backcolor);
box[0][0]=5.25;
box[0][1]=3.8125;
box[1][0]=5.3125;
box[1][1]=3.875;
box[2][0]=5.3125;
box[2][1]=3.75;
polf2(3,box);

box[0][0]=4.8125;
box[0][1]=3.375;
box[1][0]=4.8125;
box[1][1]=3.75;
box[2][0]=5.125;
box[2][1]=3.75;
box[3][0]=5.1875;
box[3][1]=3.6875;
box[4][0]=5.1875;
box[4][1]=3.4375;
box[5][0]=5.125;
box[5][1]=3.375;
polf2(6,box);

box[0][0]=4.8125;
box[0][1]=3.875;
box[1][0]=4.8125;
box[1][1]=4.125;
box[2][0]=5.125;
box[2][1]=4.125;
box[3][0]=5.1875;
box[3][1]=4.0625;
box[4][0]=5.1875;
```

```
            box[4][1]=3.9375;
            box[5][0]=5.125;
            box[5][1]=3.875;
            polf2(6,box);

            box[0][0]=5.3125;
            box[0][1]=3.75;
            box[1][0]=5.3125;
            box[1][1]=4.125;
            box[2][0]=5.125;
            box[2][1]=4.125;
            box[3][0]=5.125;
            box[3][1]=3.875;
            polf2(4, box);

            break;
        case ':':

            rectf(4.9375.3.35,5.0625,3.60);
            rectf(4.9375,3.90,5.0625,4.15);

            break;

        case ' ':

            break;

    }  /* end  switch */

}      /* end routine "letter"

/*********************************************

filename: NETV.C
author: James Manley
modified by: Michael Zyda
date: April 29, 1987

*********************************************/
```

This is the same routine used in the Navigator's Display. The code for the module can be found in Appendix A of this study.

```
/*********************************************

filename: LOADARRAY.C
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

/*

Reads road map into system array named roadmap

*/

#include "const.h"
#include "vars.ext.h"
#include "stdio.h"
#include <errno.h>
int loadarray()
{
int i;
FILE *fp;

if ((fp = fopen("roadmap","r")) == NULL)
      {
      printf("Cannot read roadmap.\n");
      return(-1);
      }
      else for (i = 0: !feof(fp); ++i)
            fscanf(fp,"%f %f %f", &roadmap[i][0], &roadmap[i][1],
                  &roadmap[i][2]);

setbell('1');
ringbell();
setbell('2');
ringbell();

return(--i);
}

/*********************************************

filename: CONST.H
```

```
author: Michael J. Dolezal
date: May 20, 1987

**********************************************/

/*

All program constants are defined in this file

*/

typedef float Dimension;

#include "gl.h"
#include "device.h"
#include "math.h"
#include "time.h"
#include "stdio.h"

#define SYSTEM_ORDER 4
#define MOUNTAIN      8
#define MOUNTAIN1     9
#define SKY          10
#define FIELD        11
#define WARN         12

#define WALL         13
#define SIDEWALL     14
#define ROOF         15
#define WINDOW       16
#define CHMWALL1     17
#define CHMWALL2     18
#define SIDEROOF     19
#define FRAME        20
#define SIDEWALL1    21
#define WALL1        22
#define ROOF1        23
#define FRAME1       24
#define WINDOW1      25
#define DIMGREEN     26
#define DIMYELLOW    27
#define DIMRED       28
#define GRAY         29
```

```
#define MAXFUEL        3000.0
#define PI             3.14
#define Crossroadlen 4000
#define REDLIGHT       1
#define YELLOWLIGHT    2
#define GREENLIGHT     3
#define ON             1
#define OFF            0
#define MPS_TO_KMPH    3.6
#define RAD_TO_DEG     57.2215      /* This is equivalent to (360)/(2 * PI) */
#define BENDRADIUS     76.0
#define ROADWIDTH      16.0
#define ROADLEN        400.0
#define LAPDIST        2174
#define DrManual       0
#define ASteerNSp      1
#define CruiseNavSteer 2
#define AUTOPILOT      3
#define CruiseDrSteer 4
#define ASteerDrSp     5
#define NavManual      6
#define DrSteerNavSp   7
#define NavSteerDrSp   8
#define BRAKEX         397
#define BRAKEY         10
#define CMDX           577
#define CMDY           10
#define BRAKEGAIN      0.0015

/*********************************************

filename: VARS.H
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

/*

All global variables are in this file.

/*
```

```
Tag transl4, transl3, transl2;
Tag transl1, transl, trans22;
Tag odotag1, odotag2, odotag3, odotag4;
Tag dangertag, temptag, belttag, braketag;
Tag fuel1, roadlooktag, skylooktag, stopsigntag;
Tag greenlighttag, yellowlighttag, redlighttag;
Tag steerwheeltag, terrain1looktag;
Tag manbraketag, manspeedtag;
Tag houselooktag, housetranstag;
Tag housescaletag;
Tag house1looktag, house1transtag;
Tag house1scaletag;

Coord latri[3][2], ratri[3][2];
float fuelbar,speedbar;
float fuelquant     = MAXFUEL;  /* Maximum fuel available      */
float heading_xpos = 429.5;     /* Heading indicator position */
float speedinc      = 1.0;      /* Speed increment/decrement  */

Device keypressed;

Boolean start       = FALSE;    /* Start of program flag       */

/*

   Larger turning_response_gain corresponds to "stiff"
   steering and lower value corresponds to "sloppy"
   steering.  Large velocity_gain corresponds to sedan
   automobile and smaller value corresponds to sport car.

   Operator has control over steer_wheel_angle and speed
   using the mouse.

   Car_time is the integration timer.

*/

float state_vector[5];
float velocity_time_consant = 9.0;
float turning_response_gain = 0.02;
float heading_angle_gain    = 0.294;
float heading_angle_rate_gain = 0.828;
float steer_wheel_angle     = 0.0;      /* Unit is radian */
```

```
float prediction_time        = 1.17;       /* Unit is second */
float sigma_dot               = 0.0;
float steer_inc                  = 0.10;       /* Unit is radian */
float car_time               = 0.0;
float deltat                 = 0.17;
float speed                   = 0.0;
float sigma                      = 0.0;


/* IRIS allow such a large array only if it is global */
float roadmap[5000][3];

Angle      Fov           = 1000;     /* Field of view 100 deg */

int distance = 0;                    /* distance traveled */


*********************************************/

filename: VARS.EXT.H
author: Michael J. Dolezal
date: May 20, 1987

*********************************************/

/*

All external variables are in this file

*/

extern Tag transl4, transl3, transl2;
extern Tag transl1, transl, trans22;
extern Tag odotag1, odotag2, odotag3, odotag4;
extern Tag dangertag, temptag, belttag, braketag;
extern Tag fuel1, roadlooktag, skylooktag, stopsigntag;
extern Tag greenlighttag, yellowlighttag, redlighttag;
extern Tag steerwheeltag, terrain1looktag;
extern Tag manbraketag, manspeedtag;
extern Tag houselooktag, housetranstag;
extern Tag housescaletag;
extern Tag house1looktag, house1transtag;
extern Tag house1scaletag;
```

```
extern Coord latri[3][2], ratri[3][2];

extern float fuelbar,speedbar;
extern float fuelquant;
extern float heading_xpos;
extern float speedinc;

extern Device keypressed;

extern Boolean start;

extern float state_vector[5];
extern float velocity_time_consant;
extern float turning_response_gain;
extern float heading_angle_gain;
extern float heading_angle_rate_gain;
extern float steer_wheel_angle;
extern float prediction_time;
extern float sigma_dot;
extern float steer_inc;
extern float car_time;
extern float deltat;
extern float speed;
extern float sigma;

extern float roadmap[5000][3];

extern Angle      Fov;

extern int distance;

/*********************************************

filename: MAP.C
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

/*

This module works independently from the rest of the system.
```

```
It generates the road map for autonomous navgiation.
Original by Tan Chiam Huat, modified by Mike Dolezal to complete the
circuit with the car in the right lane.

*/

#include <stdio.h>
#include <math.h>

main()
{
FILE *fp;
int i;
                    /*  Road Specification */
                    /* Note: Must match that used in the carsimu.c program */

float bendradius = 76.0;
float roadwidth = 16.0;
float len1 = 400.0;
float len2 = 400.0;
float len3 = 400.0;
float len4 = 400.0;
float newx, newy, miss;
float calx, caly, start_rad;
float perstep_rad;

                    /* road map increment step */
float step = 1.0;
float rad1 = bendradius;
float rad2 = bendradius;
float rad3 = bendradius;
float rad4 = bendradius;
float lastxvalue;
float lastyvalue;
float x1, y1, z1;
float x2, y2, z2;
float x3, y3, z3;
float x4, y4, z4;
float x5, y5, z5;
float x6, y6, z6;
float x7, y7, z7;
float x8, y8, z8;
                    /* Road Segment Specifications */
```

```
x1 = 0.0; y1 = 0.0; z1 = 0.0;
x2 = 0.0; y2 = len1; z2 = 0.0;
x3 = rad1; y3 = y2 + rad1; z3 = 0.0;
x4 = x3 + len2; y4 = y3; z4 = 0.0;
x5 = x4 + rad2; y5 = y4 - rad2; z5 = 0.0;
x6 = x5; y6 = y5 - len3; z6 = 0.0;
x7 = x5 - rad3; y7 = y6 - rad3; z7 = 0.0;
x8 = x7 - len4; y8 = y7; z8 = 0.0;

fp = fopen("roadmap","w");

newy = y1;
for (i = 0; newy <= y2; ++i)
     {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",x1,newy,z1);
#endif

       fprintf(fp,"%.2f %.2f %.2f\n",x1,newy,z1);
       lastyvalue = newy;
       newy += step;
       }
newy = lastyvalue;
miss = y2 - newy;

#ifdef DEBUG
   printf("miss1 %.2f\n",miss);
#endif
start_rad = 0;
if (miss > 0)
     {
     start_rad = miss/rad1;
     calx = cos(start_rad);
     caly = sin(start_rad);
     newy += caly;
     newx += calx;

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",x2+newx,y2+newy,z2);
#endif

     fprintf(fp,"%.2f %.2f %.2f\n",x2+newx,y2+newy,z2);
```

```
    }

perstep_rad = step/rad1;
for (i = 0; newx <= x3; ++i)
    {
    start_rad += perstep_rad;
    calx = rad1 * cos(start_rad);
    caly = rad1 * sin(start_rad);
    lastxvalue = newx;
    lastyvalue = newy;
    newy = y2 + caly;
    newx = x2 + (rad1 - calx);
    if (newx < x3)
       {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z2);
#endif

       fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z2);
       }
    }

newx = lastxvalue;
newy = lastyvalue;
miss = x3 - newx;

#ifdef DEBUG
   printf("miss2 %.2f\n",miss);
#endif

if (miss > 0)
    {
    newx = x3 + miss;

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,y4,z3);
#endif

     fprintf(fp,"%.2f %.2f %.2f\n",newx,y4,z3);
     }

for (i = 0; newx <= x4; ++i)
```

```
    {
    lastxvalue = newx;
    newx += step;
    if (newx <= x4)
        {
#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,y4,z3);
#endif

        fprintf(fp,"%.2f %.2f %.2f\n",newx,y4,z3);
        }
     }
newx = lastxvalue;
miss = x4 - newx;

#ifdef DEBUG
   printf("miss3 %.2f\n",miss);
#endif

start_rad = 0;
if (miss > 0)
    {
    start_rad = miss/rad2;
    caly = rad2 * cos(start_rad);
    calx = rad2 * sin(start_rad);
    newy = y4 - (rad2 - caly);
    newx = x4 + calx;

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z4);
#endif

    fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z4);
    }

perstep_rad = step/rad1;
for (i = 0; newy >= y5; ++i)
     {
     start_rad += perstep_rad;
     caly = rad2 * cos(start_rad);
     calx = rad2 * sin(start_rad);
     lastxvalue = newx;
     lastyvalue = newy;
```

```
      newx = x4 + calx;
      newy = y4 - (rad2 - caly);
      if (newy >= y5)
          {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z4);
#endif

          fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z4);
          }
      }

newx = lastxvalue;
newy = lastyvalue;
miss = newy - y5;

#ifdef DEBUG
   printf("miss4 %.2f\n",miss);
#endif
if (miss > 0)
     {
     newy = y5 + miss;
#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
     fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
     }
for (i = 0; newy >= y6; ++i)
     {
     lastyvalue = newy;
     newy -= step;
     if (newy >= y6)
          {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif

          fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
          }
     }
newy = lastyvalue;
```

```
miss = newy - y6;

#ifdef DEBUG
   printf("miss5 %.2f\n",miss);
#endif

start_rad = 0;
if (miss > 0)
     {
     start_rad = miss/rad3;
     calx = rad3 * cos(start_rad);
     caly = rad3 * sin(start_rad);
     newx = x6 - (rad3 - calx);
     newy = y6 - caly;

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif

     fprintf(fp."%.2f %.2f %.2f\n",newx,newy,z5);
     }

perstep_rad = step/rad3;
for (i = 0; newx >= x7; ++i)
      {
      start_rad += perstep_rad;
      calx = rad3 * cos(start_rad);
      caly = rad3 * sin(start_rad);
      lastxvalue = newx;
      lastyvalue = newy;
      newy = y6 - caly;
      newx = x6 - (rad3 - calx);
      if (newx >= x7)
           {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
           fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
           }
      }

newx = lastxvalue;
```

```
newy = lastyvalue;
miss = newx - x7;

#ifdef DEBUG
   printf("miss6 %.2f\n",miss);
#endif

if (miss > 0)
     {
     newx = x7 - miss;

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,y8,z8);
#endif

     fprintf(fp,"%.2f %.2f %.2f\n",newx,y8,z8);
     }

for (i = 0; newx >= x8; ++i)
     {
     lastxvalue = newx;
     newx -= step;
     if (newx >= x8)
          {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,y8,z8);
#endif

          fprintf(fp,"%.2f %.2f %.2f\n",newx,y8,z8);
          }
     }
newx = lastxvalue;
miss = newx - x8;

#ifdef DEBUG
   printf("miss7 %.2f\n",miss);
#endif

                    /* Process curve #4   */

newx = lastxvalue;
newy = lastyvalue;
```

```
miss = x8 - newx;

#ifdef DEBUG
   printf("miss8 %.2f\n",miss);
#endif

start_rad = 0;
if (miss > 0)
     {
     start_rad = miss/rad4;
     caly = rad4 * cos(start_rad);
     calx = rad4 * sin(start_rad);
     newy = y8 + (rad4 - caly);
     newx = x8 - calx;
#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z4);
#endif

     fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z4);
     }
perstep_rad = step/rad1;
for (i = 0; newy <= y1; ++i)
     {
     start_rad += perstep_rad;
     caly = rad4 * cos(start_rad);
     calx = rad4 * sin(start_rad);
     lastxvalue = newx;
     lastyvalue = newy;
     newx = x8 - calx;
     newy = y8 + (rad4 - caly);
     if ((newy >= y8) && (newy <= y1))
          {

#ifdef DEBUG
   printf("%.2f %.2f %.2f\n",newx,newy,z8);
#endif

          fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z8);
          }
     }
fclose(fp);
}  /* main */
```

```
/*********************************************

filename: MAKEFILE
author: Tan Chiam Huat
modified by: Michael J. Dolezal
date: May 20, 1987

*********************************************/

CFLAGS = -Zf
SRCS = other.c\
       netV.c\
       integrate.\
       display.c\
       letter.c\
       welcome.c\
       find_subgoal.c\
       checkkey.c\
       loadarray.c\
       circuit.c\
       carsimu.c

OBJS = other.o\
       netV.o\
       integrate.o\
       display.o\
       welcome.o\
       carsimu.o\
       find_subgoal.o\
       checkkey.o\
       loadarray.o\
       circuit.o\
       letter.o
carsimu: $(OBJS)
       cc -o carsimu $(OBJS) -Zf -Zg -lm -lbsd -ldbm

$(OBJS): const.h
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. Robert B. McGhee, Code 52Mz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	20
4.	Dr. Michael J. Zyda, Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Center for Naval Analyses 2000 N. Beauregard Street, Alexandria, Virginia 22311	1
6.	Director of Research Administration Code 012 Naval Postgraduate School Monterey, California 93943	1
7.	Mr. Russel Davis HQ, USACDEC Attention:ATEC-IM Fort Ord, California 93941	2
8.	LCDR H. R. Everett Naval Oceans Systems Center, Code 442 San Diego, California 92152	1

9. Dr. William E. Isler — 1
Defense Advanced Research Projects Agency/ISTO
1400 Wilson Blvd.
Arlington, Virginia 22209

10. Dr. Andrew Chang — 1
Central Engineering Laboratories
FMC Corporation
1185 Coleman Ave, Box 580
Santa Clara, California 95052

11. Dr. James Lowrie — 1
Mail Stop T0427
Martin Marietta Denver Aerospace
P.O. Box 179
Denver, Colorado 80201

12. Dr. D. Y. Tseng — 1
Artificial Intelligence Center
Hughes Research Laboratories
23901 Calabasas Rd.
Calabasas, California 91302-1579

13. Prof. R. E. Fenton — 1
Dept. of Electrical Engineering
Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

14. Prof. K. W. Olson — 1
Dept. of Electrical Engineering
Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

15. Major Michael J. Dolezal — 4
1322 Rue Crozat
Baton Rouge, Louisiana 70810

16. Chief of Naval Operations — 1
Director, Information Systems (OP-945)
Navy Department
Washington, DC 20350-2000

17. Mr. Doug Gage 1
Naval Oceans Systems Center, Code 442
San Diego, California 92152

18. Chairman 1
Computer Science Department, Code 52
Naval Postgraduate School
Monterey, California 93943-5000

19. Curricular Officer 1
Computer Technology Programs, Code 37
Naval Postgraduate School
Monterey, California 93943-5000

Thesis
D6411
c.1
Dolezal
A simulation study of a speed control system for autonomous on-road operation of automotive vehicles.